2

COMPUTER-AIDED ALGORITHM DESIGN AND REARRANGEMENT

- Michele M. Covell
  SRI International
- Cory S. Myers
  Lockheed Sanders, Inc.
- Alan V. Oppenheim
  Massachusetts Institute of Technology

2.1 INTRODUCTION

The design of signal processing systems typically involves a high-level specification of the requirements of the system, development of an appropriate algorithm or set of algorithms to accomplish these requirements, and the implementation of the algorithms in an appropriate technology. Often these stages are not independent, and in particular, the detailed structure of the algorithms for implementing the system needs to take into account a variety of cost measures and options relating to system requirements, such as speed, modularity, and so on, and a variety of architectural constraints or technologies available for the implementation. For these reasons, it is important that algorithm design explore a wide variety of implementations which are found by exploiting the underlying mathematics of signal processing and taking into account a variety of cost measures. This is usually done by a design engineer with a detailed knowledge of and insight into a variety of transformations. From an input/output point of view, these transformations result in equivalent signal processing operations, but may have very different implications with regard to various cost measures.

In carrying out signal processing system design, there are a few design tools currently available, but these primarily provide a convenient environment for writing

programs or for testing algorithms on data. There are no systems available presently that remap signal processing algorithms, specified at a high level of abstraction, to algorithmic descriptions that are more efficient in the context of particular implementational or architectural constraints. Such a design environment, if successful, might generate designs that improve on those that would be generated by experienced systems designers. A more likely and also highly desirable outcome, at least in the short run, is an environment that, with modest human intervention, achieves fast designs of signal processing algorithms that are reasonably efficient in relation to hand designs by sophisticated systems designers. The potential ability to do this in a signal processing context stems from the fact that, for signal processing, algorithmic transformations tend to follow a clearly defined set of mathematical rules. Based on these rules, the space of equivalent algorithms can, in principle, be explored to determine those that are most efficient using appropriate cost measures.

This chapter attempts to demonstrate both the feasibility and the advantages of a signal processing design environment which incorporates symbolic algorithm manipulation along with numerical processing. In particular, we focus on the problem of developing automated tools to support algorithm manipulation. As viewed in this chapter, algorithm manipulation includes property and transform analyses and algorithm rearrangement. Property and transform analyses provide information about an algorithm or its output signal. For example, determination that the output of an FFT will be conjugate symmetric because its input is real, is property analysis. Some examples of properties that are widely used in signal processing are computational cost, stability, causality, symmetry, linearity, and time-invariance. The use of a z-domain representation of a linear, time-invariant system to determine stability is an example of transform analysis. An example of algorithm rearrangement is shown in Figure 2.1. Figure 2.1(a) describes one implementation of a noninteger sampling rate conversion: upsample by five, low-pass filter, and downsample by four. This simple description of a noninteger sampling rate conversion is easily designed and implemented but is less computationally efficient than the implementations shown in Figures 2.1(b) and (c), which also provide 4 : 5 noninteger sampling rate conversions. The desired capabilities of an environment that integrates algorithm definition, manipulation and analysis, along with numerical processing, are discussed in section 2.3, along with the constraints that these capabilities impose.

In order to demonstrate the advantages of a design environment that combines symbolic algorithm manipulation with numerical processing, two application areas are considered: noninteger sampling rate conversion and code-division sonar imaging. Both of these application areas are discussed in section 2.2. Algorithms for these applications are defined and manipulated in ADE [1]. ADE (A Design Environment), described in more detail in section 2.4, is an experimental environment that shows the feasibility of a signal processing workstation with integrated tools for the specification of algorithms, computer-aided analysis, and manipulation of those algorithms, as well as application of the algorithms to numerical sequences.

ADE is based on E-SPLICE [2], the first system to demonstrate automated property analysis and algorithm manipulation for digital signal processing. ADE and

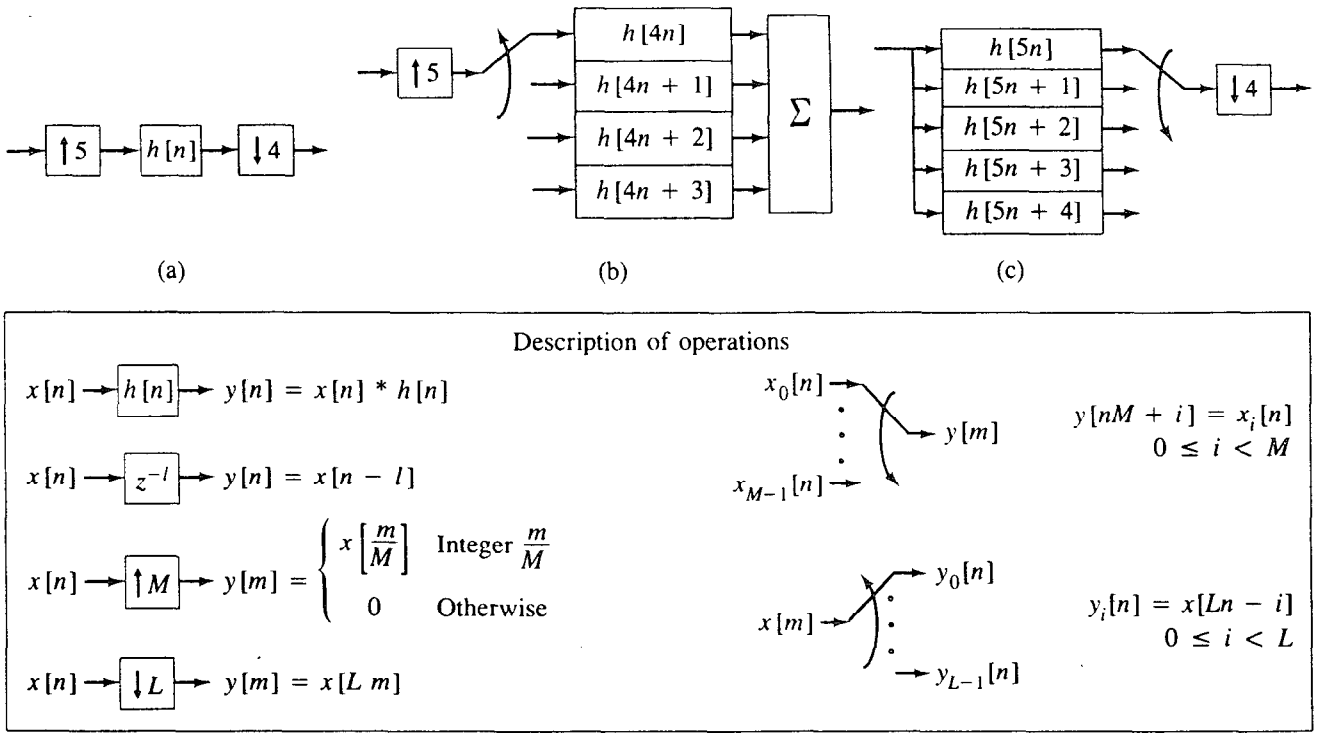


Figure 2.1 Alternate implementations of a 4 : 5 sampling rate conversion.

E-SPLICE are the results of a sequence of research efforts into integrated signal processing environments. These research efforts started in the late 1970s with Kopec's development of SRL [Chapter 1 this volume; 3]. SRL provides data abstractions that both reflect the basic characteristics of signals and support numeric manipulations. SPLICE [2, 4] resulted from an effort to improve the computer representation of signals beyond the work that had already been done by Kopec. Subsequently, with E-SPLICE, Myers [2] expanded the scope of the software environment to include symbolic manipulation of algorithms in addition to the previously supported numerical manipulation of signals. ADE [this chapter; 1] refines the symbolic rearrangement capabilities of E-SPLICE, making possible the manipulation of large signal processing expressions.

2.2 SIGNAL PROCESSING TOPICS USED IN EXAMPLES

In this chapter, two examples, noninteger sampling rate conversion and sonar FSK-code detection, are used to illustrate the potential of an integrated signal processing environment that combines algorithm definition, manipulation, and analysis with numerical processing. These applications are described in this section.

2.2.1 Noninteger Sampling Rate Conversion

Discrete-time sequences are often used to represent a bandlimited, continuous-time signal. Often, it is desirable to change the sampling rate that defines the conversion between discrete and continuous time. For example, film is shot at 24 frames/sec.

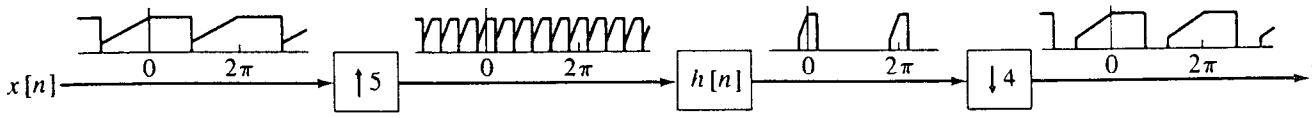


Figure 2.2 Effect of a 4 : 5 sampling rate conversion in the Fourier domain.

To display the film at the correct rate on American television, the temporal sampling rate must be changed to 30 frames/sec.[1] This 4 : 5 sampling rate conversion can be implemented using the structure shown in Figure 2.1(a). The effect of these operations in the frequency domain is shown in Figure 2.2. The low-pass filter between the upsampling and downsampling operations is necessary to prevent temporal aliasing. Unfortunately, this filter operates at a high data rate: for each four input points and five output points of the overall system, twenty input and output points pass through the low-pass filter. Figures 2.1(b) and (c) show two alternate implementations in which the filter runs at lower rates. Figure 2.1(b) shows a polyphase implementation of the filter/downsample part of the sampling rate conversion: the four, shorter filters each operate at one fourth the rate of the filter in Figure 2.1(a) and, if the convolutions are implemented directly, the computational requirements are reduced by a factor of four from the original implementation. Figure 2.1(c) shows a polyphase implementation of the upsample/filter part of the sampling rate conversion: the five, shorter filters each operate at one fifth the rate of the original filter and, again assuming direct implementation of the convolutions, the computational requirements are reduced by a factor of five.

The two polyphase implementations shown in Figure 2.1(b) and (c) are well documented in multirate filtering literature [5]. Another polyphase implementation for noninteger sampling rate conversion, which has only recently been documented in signal processing literature, was generated by E-SPLICE [2]: this alternate implementation will be introduced and discussed in section 2.5.

2.2.2 Modulated Filter Banks and Short-Time Fourier Transforms

Conventional sonar imaging systems achieve spatial resolution either through the use of a single, swept beam or through the use of multi-element arrays. These techniques, while highly successful, present some inherent difficulties. In the case of the single swept beam, the time required to scan through the desired aperture can result in the failure to detect transients. When multi-element arrays are used instead, the hardware requirements necessary to achieve high resolution can result in a large, costly system. Jaffe and Richardson [6] propose an alternative to these two techniques using the simultaneous transmission of a set of coded waveforms.

The transmitter in the proposed system is a set of N transducers, each illuminating a different direction and each transmitting a distinct signal, S_i for

[1] Each frame in American television is made up of two vertically interlaced fields. Sixty fields are displayed per second.

$i = 0, \ldots, N-1$. One wide-beam hydrophone is used as a receiver. Multiple-hypothesis testing is then used to detect and discriminate the returns from the separate beams. In order to achieve good spatial resolution, the set of signals $\{S_0, \ldots, S_{N-1}\}$ must have good signal-to-signal rejection for all possible time delays. In addition, to achieve good range resolution, each signal should have a sharply peaked autocorrelation function. Jaffe and Richardson [6] propose a specific set of FSK codes with these properties:

$$S_i(t) = \sum_{k=0}^{N-1} Re\{P_{i,k}(t)e^{-j2\pi f_c t}\} \qquad \text{for } i = 0, \ldots, N-1$$

$$P_{i,k}(t) = C_l(t - kT)$$

where $l = p_i(k)$ provides, for each i, a different permutation of the numbers $0, \ldots, N-1$ versus k and

$$C_l(t) = w(t)e^{-j(2\pi/T)lt}$$

where $w(t) = 0$ for $t < 0$ and for $t \geq T$

Specifically, each signal is made up of a sum of N individual, uniformly spaced frequency bursts (commonly referred to as frequency chips). When $N + 1$ is prime, $p_i(k)$ can be selected such that the signals and all their circular shifts achieve maximal Hamming distance separation.[2] The window $w(t)$ allows the frequency chips to be shaped to adjust their side-lobe characteristics.

The received signal can be modeled as a superposition of the reflected energy from each of the illuminated scattering centers:

$$r(t) = \sum_{i=0}^{N-1} \sum_{m=1}^{M_i} \rho_{i,m} \sum_{k=0}^{N-1} Re\{e^{j\varphi_{i,m,k}} P_{i,k}(t - \tau_{i,m})e^{-j2\pi f_c(t-\tau_{i,m})}\}$$

The summation over i represents the superposition of the returns from different transmitter beams, and the summation over m represents the superposition of the returns from the M_i scattering centers within the ith beam. $\rho_{i,m}$ is a positive number representing the strength of the return from the mth scattering center in the ith transmitter beam: it is determined by the scattering cross section and the distance of the target. $\tau_{i,m}$ is the propagation delay for the combined forward and return paths to and from the scattering center. $\varphi_{i,m,k}$ represents a nonuniform phase distortion on the kth chip of the FSK code introduced by the scattering characteristics of the target and by the fluctuations in the propagating medium. The possibility of a Doppler frequency shift is ignored in this model.

From this model, the values of $\rho_{i,m}$ and $\tau_{i,m}$ as a function of i and m provide the desired sonar "image." For any given time delay, $\rho_{i,m}$ can be estimated using a detector that minimizes the mean square error. Estimation of $\tau_{i,m}$ can be avoided by simply estimating $\rho_{i,m}$ for all resolvable time delays.[3] Using this approach, the

[2] The Hamming separation distance is the minimum number of elements that differ between a code word and any of the circular shifted or unshifted versions of another code word.

[3] Since the total bandwidth of the transmitted signal is N/T Hz, the resolvable time delay is approximately T/N seconds.

discrete-time approximation to the detectors is shown in Figure 2.3. In-phase and quadrature samples are taken of the received signal after demodulation by the carrier frequency, f_c. Matched filters are used to detect the individual frequency chips. Since the model allows for an unknown, nonuniform phase distortion between frequency chips, incoherent summation is used between frequency chips: this incoherent combination is completed in the last box in Figure 2.3. Finally, to avoid à priori estimation of $\tau_{i,m}$, the output from these detectors is computed at each point in time.

The digitized frequency chips will be $w(nT/N)e^{-j(2\pi/N)kn}$ for $k = 0, \ldots, N-1$. Thus, the outputs from the frequency-chip matched filters are:

$$y_k[t] = x[t] * \left(w\left(-t\frac{T}{N}\right)e^{j(2\pi/N)kt} \right)$$

$$= \sum_{n=-(N-1)}^{0} x[t-n]w\left(-n\frac{T}{N}\right)e^{j(2\pi/N)kn} \tag{2.1}$$

$$= \sum_{n=0}^{N-1} x[t+n]w\left(n\frac{T}{N}\right)e^{-j(2\pi/N)kn} \tag{2.2}$$

From (2.1), the frequency-chip matched filters can be implemented as a modulated filter bank.

A well-known implementation of a modulated filter bank is the short-time Fourier transform (STFT). By defining $v_t[n] = x[t+n]w(nT/N)$, (2.2) can be seen to be the N-point discrete Fourier transform (DFT) of $v_t[n]$. Thus, the matched filters can be implemented using a STFT. Formulating the matched filters as a STFT allows the use of well-known, computationally efficient implementations of the STFT. The most obvious of these is the FFT: an N-point FFT of $v_t[n]$ can be separately computed at each time sample t. This approach requires $O(N \log N)$ computations per time sample t and allows temporally sparse computation of $y_k[t]$ without any increase in complexity per output point.

A third implementation of the frequency-chip matched filters can be derived by again considering (2.2). Defining

$$x_t[n] = \begin{cases} x[t+n] & 0 \le n < N \\ 0 & \textit{otherwise} \end{cases}$$

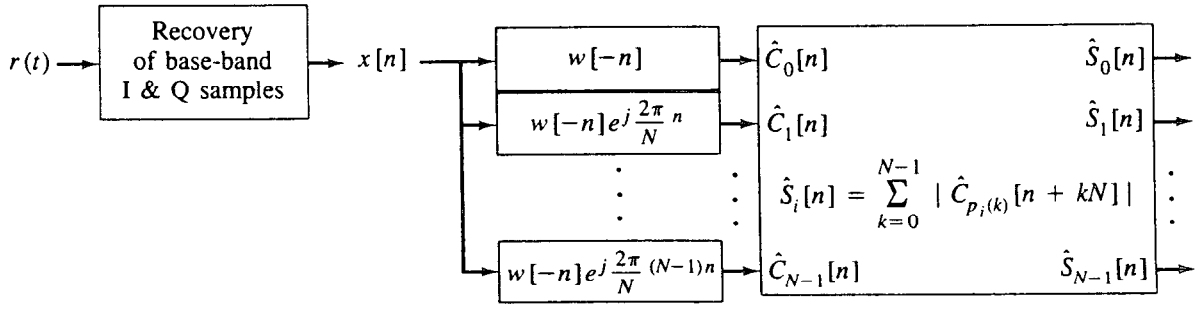


Figure 2.3 The discrete-time approximation to the optimal detectors for N FSK-coded sonar signal beams.

with $X_t[k]$ and $W[k]$ as the N-point DFT's of $x_t[n]$ and $w(nT/N)$, $y_k[t] = (1/N)X_t[k] \circledN W[k]$ where $\circledN$ is an N-point circular convolution. This approach imposes the shaping provided by $w(t)$ through frequency-domain convolution. The advantage of shaping $x_t[n]$ after computing the DFT is that the Goertzel algorithm can then be used. In particular

$$X_{t+1}[k] = e^{j(2\pi/N)k}(X_t[k] + x[t+N] - x[t])$$

Using this recursive approach to find $X_t[k]$ requires $O(N)$ computations per time sample t. The total computational cost of computing $Y_k[t]$ depends on the form of $W[k]$: if $W[k]$ has only a few nonzero samples, the additional computational cost imposed by shaping may be much lower than would be required by a general convolution of two sequences. A disadvantage of this approach is a problem with stability: the recursive computation depends on pole/zero cancellation on the z-domain unit circle.

As will be described in section 2.4, ADE generated a fourth, innovative implementation of the frequency-chip matched filters [1]. This new approach, which will be referred to as the pruned FFT, could not be found in either modulated filter bank or short-time Fourier transform literature. Like the recursive computation of the STFT, the pruned FFT imposes the shaping provided by $w(t)$ through frequency-domain convolution and, like the recursive STFT, only $O(N)$ computations per time sample t are required to compute $X_t[k]$. The pruned FFT has the advantage over the recursive STFT of unconditional stability.

2.3 CAPABILITIES AND REQUIREMENTS FOR AN INTEGRATED SIGNAL PROCESSING ENVIRONMENT

Section 2.1 asserted that a signal processing workstation that provides an integrated environment for numerical processing, algorithm definition, and algorithm manipulation could potentially simplify the design of new signal processing algorithms, as well as improve the reliability of the design process. Supporting both algorithm definition and numeric processing allows the engineer to test the behavior of algorithms as soon as they are defined. Similarly, supporting both algorithm definition and property/transform analysis would allow analytical tools to be easily applied by computer, such as studying the z-domain representations of linear time-invariant systems. By including algorithm rearrangement in the workstation's capabilities, a high-level signal processing "compiler" can be used to apply well-known global transformations to an initial description of an algorithm. One example of this high-level compilation would be the transformation from the noninteger sampling rate conversion shown in Figure 2.1(a) to one of the polyphase structures shown in Figures 2.1(b) and (c). Another possible advantage to a high-level signal processing compiler is the derivation of new, computationally efficient implementations for the given signal processing operation. Two examples of new, innovative implementations found by high-level signal processing compilers will be discussed later in this chapter.

Given that our goal is to expedite signal processing algorithm design, some effort should be made to determine what capabilities are desired and what constraints are imposed by these desired capabilities. A general statement of the desired capabilities has already been made: to support numeric processing of signals, algorithm definition, signal and system property analysis, and algorithmic rearrangement, all in a consistent, well-integrated manner. Each of these requirements in turn imposes constraints on the signal representation that is chosen. Some of these constraints have been pointed out by Kopec in Chapter 1. In this section, we review the requirements described by Kopec and we introduce other requirements, to allow for algorithm manipulation and analysis.

2.3.1 Support for Numerical Signal Processing

Providing consistent, well-formulated support for numeric processing of signals and for algorithm definition imposes some basic constraints on the representation of signals, some of which were discussed in Chapter 1. The signal representation should be explicit and unique, with uniform external behavior; it should distinguish between the domain and the nonzero support of the signal; and it should be externally immutable. The same signal representation should be used for both numerical processing and for the signal analysis and manipulation operations. In more detail:

- **Explicit, unique signal identity:** In signal processing, signals are not just an ordered collection of sample values, but instead have a unique identity and inherent properties of their own [Chapter 1]. Many of their properties, such as nonzero support, domain, and symmetry, are closely tied to the sample values, but others, like the algorithm that generated the signal or its computational cost, cannot be derived from the sample values. Therefore, an explicit signal representation, distinct from a simple ordered set of sample values, is necessary.
- **Uniform external interface to signals:** To simplify the interface of signals and systems, signal representations should all have the same external behavior, independent of the internal computational method used to generate the sample values of the signal [2, 4]. For example, the retrieval of a sample value from a point operator, such as a multiplication block, should externally appear the same as the retrieval of an array operator, such as an FFT block, and the same as the retrieval from a state-machine operator, such as an IIR filter.
- **Distinct signal domain and nonzero support:** The signal representation should distinguish between the domain and the nonzero support of the signal [2, 4]. The domain of a signal determines where the signal is defined: discrete-time sequences are defined on all integer time indices and undefined elsewhere; discrete-time Fourier transform signals are defined on all real frequency indices; and z-transform signals are defined on the annulus of complex indices inside its region of convergence and undefined elsewhere. Any sample value within the defined domain of the signal should be accessible. Accessing a signal inside its domain but outside its nonzero support should return the sample value of the signal at that point, namely zero.

• **Immutable signal representation:** An additional constraint on a signal representation for numerical processing is that signals should be immutable, i.e., there is no operation that can change the properties of a signal once it is defined. Mathematically, signals are immutable objects: their identity and their properties are fixed and unchanging. For example, the sample values, symmetry, and nonzero support of the 256-point Hanning window are completely defined and immutable. Using this sequence as input to a system does not alter the sequence, but instead produces a new sequence. As pointed out in Chapter 1, this immutability in signals also simplifies and clarifies the signal processing algorithms that use them: immutability makes signals referentially transparent.

2.3.2 Support for Algorithm Definition

Supporting algorithm definition constrains the internal representation of signals and systems, just as the support of numerical processing of signals constrains the external behavior of signals. Besides the constraints already discussed, algorithm definition is simplified if there are multiple computational models that are supported by the signal processing environment. Four computational methods that are widely used in signal processing are point operations, array operations, state-machine models, and composition operations.

- **Array operators**, such as FFTs, compute multiple sample values simultaneously.
- **State-machine models** generate sample values sequentially using an internal state vector: an IIR filter, for example, could be easily implemented using a state-machine model.
- **Point operations**, in contrast to array operators and state-machine models, generate one sample value at a time in any random order. Two examples of point operations are addition or multiplication.
- **Composition operations** are implicitly defined through the cascade or "composition" of other, previously defined systems. An example would be the definition of a cosine sequence as the sum of two complex exponentials.

Supporting all four of these computational methods simplifies the programmer's task, since algorithm definitions do not have to be forced into a computational form that is ill-suited for the operation at hand. Providing multiple computational methods while maintaining a uniform external interface decouples the internal and external characteristics of the signal [2, 4]. Thus, the user of a signal need not be concerned with the computational method that the programmer of the signal chose.

2.3.3 Support for Signal Property and Transform Analysis

The premise of this chapter is that a well-integrated signal processing design environment should support property and transform analysis, as well as the more standard algorithm definition and numeric processing steps. Property analysis allows specific

questions about the characteristics of a signal or system (e.g., symmetry or linearity) to be answered. Transform analysis uses an alternate representation of a signal or system (e.g., a z-domain representation) to emphasize some aspect of the signal or system that is not apparent in the time-domain representation. The automation of property and transform analysis would allow these analytical tools to be easily and reliably applied by computer. Although property and transform analysis could theoretically be completed using "first principles," this approach to property and transform analysis would be slow and unwieldy at best. Instead, the approach that is envisioned relies on explicitly including information in the signal and system definitions about properties and transforms. This approach to the automation of property and transform analysis would require:

- **Explicit descriptions of signal properties:** For example, the definition of cosine sequences would explicitly include the fact that these sequences are real and symmetric. With this type of information, the environment could provide the values of signal properties, such as those listed in Table 2.1. Furthermore, by providing the user with the appropriate tools, additional properties (e.g., cyclostationarity) could be added by simply adding this extra information to the signal definitions.
- **Explicit descriptions of the effects of systems on signal properties:** For example, by indicating the effect of a shift operator on the symmetry of a sequence (i.e., that it shifts the point of symmetry) and on the sample value type of a sequence (i.e., no effect), the symmetry and sample value type of a shifted cosine signal could be determined by the signal processing environment. This information about a system would describe its effect on the signal properties of its inputs.
- **Explicit identification of signal transforms:** For example, the definition of rectangular-window sequences would include the fact that their discrete-time Fourier transform is an aliased sinc signal. With this information, the environment could provide closed-form expressions for signal transforms, such as discrete-time Fourier transforms (DTFTs) and z-transforms.
- **Explicit descriptions of the effects of systems in the signal transform space:** For example, by indicating the effect of a shift operator on the Fourier-domain representation of its input (i.e., modulation by a complex exponential signal), the Fourier-domain representation of a shifted rectangular window could be determined by the signal processing environment. This information about a system would describe its effects on the DTFT and z-domain representations of its inputs.

TABLE 2.1 SOME USEFUL SIGNAL AND SYSTEM PROPERTIES

INVERTIBLE-P, INVERSE-SYSTEM: Whether or not the system is invertible and if so, its inverse.
SAMPLES-COMPUTABLE-P, COMPUTABLE-P: The computability of individual sample values of the signal and of all the sample values of the signal.
SAMPLE-TYPE: The data type of the sample values of the signal.
RANGE: The range of the sample values of the signal.
NON-ZERO-SUPPORT: The indices on which the sample values of the signal may be nonzero.
PERIODICITY: The repetition period of the signal.
SYMMETRY: The description of the symmetry characteristics of the signal.

2.3.4 Automation of Algorithm Rearrangement

One of the most desirable capabilities for a signal processing environment is the automation of algorithm rearrangement. This would in effect provide the engineer with a high-level signal processing "compiler." This compiler could be used both to apply well-known global transformations to an algorithm and to derive customized, computationally efficient implementations for unusual algorithms. High-level compilation of algorithms requires both the ability to enumerate mathematically equivalent implementations and the ability to compare these alternate implementations to determine their relative merit, based on computational efficiency [1, 2].

Enumeration of Mathematically Equivalent Implementations

Since high-level compilation of an algorithm relies on enumeration of mathematically equivalent implementations, the distinction between mathematical equivalence and computational equivalence is important in algorithm manipulation. *Mathematical equivalence*, or equality, between signals implies that the domain and all of the sample values of the signals are equal, even though the signals may have used different computations to arrive at those sample values. An example of mathematical equivalence is provided by the 256-point DFT of the 256-point Hanning window[4] and the sequence $\frac{1}{2}\delta[k] - \frac{1}{4}\delta[k-1] - \frac{1}{4}\delta[k-255]$. Assuming infinite computational precision, the domain and the sample values of the sequences are equal, even though they were arrived at through very different paths. *Computational equivalence* between signals implies that *all* the signal's properties are identical. This includes the sequence of computations used to arrive at the signal sample values (i.e., the generating system or algorithm). Thus, the only way to get to computationally equivalent signals is to use the same input signals into the same sequence of operations: the output of an FFT operator applied to a discrete-time impulse is computationally equivalent only to the output from the same FFT operator applied to an identical discrete-time impulse. Computational equivalence may seem like a tautology, involving statements like "$\sin x = \sin x$," but it is an equivalence that is often lost in computer programming languages. For example, in most programming languages, the mathematical equality of two copies of the output from an FFT operator would be difficult to determine, requiring a sample-by-sample comparison of the two output sequences, and computational equality would not be determinable based on these output samples.

In algorithm rearrangement, mathematical equivalence must be maintained

[4] An M-point Hanning window, $w[n]$, is [7]

$$w[n] = \begin{cases} \frac{1}{2} - \frac{1}{2}\cos(2\pi n/M) & 0 \le n < M \\ 0 & \textit{otherwise} \end{cases}$$

even though computational equivalence is deliberately lost. To provide these lists of mathematically equivalent (but computationally distinct) signals requires:

- **Explicit identification of equivalent signals:** For example, by noting the mathematical equivalence between the Hanning window sequence and a raised, windowed cosine sequence, a list of two mathematically equivalent signals could be collected.

- **Recursive identification of equivalent signals and identification and substitution of equivalent subexpressions:** This requirement is most easily explained by example. Using the previous example of the Hanning window, assume that another mathematical equivalence was noted between the cosine sequence ($\cos \omega n$) and the sum of the conjugate pair of complex exponentials ($(e^{j\omega n} + e^{-j\omega n})/2$). The environment must be able to put these two pieces of information together and increase the list from the two equivalent sequences (the Hanning window and the raised, windowed cosine) to three (the Hanning window; the raised, windowed cosine; and the raised, windowed sum of the conjugate pair of complex exponentials). This requires that the environment search recursively for equivalent signals using newly discovered signals (in this case, the raised, windowed cosine sequence) and that the environment search for signals that are equivalent to the subexpressions of the complete signal description (in this case, the cosine sequence is a subexpression of the raised, windowed cosine). This search strategy is described in detail in sections 2.5 and 2.6.

To provide these lists of mathematically equivalent systems requires:

- **Explicit identification of equivalent output signals:** For algorithm manipulation, lists of mathematically equivalent systems must be derived from information included in the system definitions. Often, the most straightforward way to provide this information is to describe the signals that are mathematically equivalent to the output signal when the system has been applied to some general or partially specified input. Without this shift in focus, most algorithmic transformations are difficult to specify, since the input signals and parameters of the system are unbound until the system is applied. For example, the description of the Goertzel algorithm as being mathematically equivalent to one channel from the rectangularly windowed STFT is not possible without being able to refer to the input signal, $x[n]$. Thus, this approach to finding equivalent algorithms places two requirements on the environment: that the system definitions include information about signals that are equivalent to their output signal and, as discussed next, that general or partially specified signals can be represented and manipulated.

- **Representation of general or partially specified signals:** As mentioned previously, one straightforward method of finding mathematically equivalent implementations of an algorithm is to use general or partially specified signals as the inputs to the algorithm and to then find signals that are mathematically equivalent to the output signal from the algorithm: the equivalent algorithms are then given by the composition of systems used to generate the equivalent output signals. This approach to algorithm manipulation is highly reminiscent of the algebraic manipulation that engineers commonly do on signal processing equations. In algebraic manipula-

tion of signal processing expressions, the inputs are represented by an algebraic variable, such as $x[n]$, and the output signal generated by processing this input is then manipulated. For example, the derivation of the Goertzel algorithm from the direct formulation of the STFT would start with the input to the STFT being represented by $x[n]$. The output signal from the DFT would then be

$$y_k[t] = \sum_{n=0}^{N-1} x[t+n]e^{-j(2\pi/N)kn}$$

The Goertzel algorithm could then be derived by manipulating $y_k[t]$ without making any assumptions about the sample values or properties of $x[n]$:

$$y_k[t+1] = \sum_{n=0}^{N-1} x[t+1+n]e^{-j(2\pi/N)kn} \tag{2.3}$$

$$= e^{j(2\pi/N)k} \sum_{m=1}^{N} x[t+m]e^{-j(2\pi/N)km} \tag{2.4}$$

$$= e^{j(2\pi/N)k} \sum_{m=0}^{N-1} x[t+m]e^{-j(2\pi/N)km} + e^{j(2\pi/N)k}x[t+N] - e^{j(2\pi/N)k}x[t] \tag{2.5}$$

$$= e^{j(2\pi/N)k}(y_k[t+1] + x[t+N] - x[t]) \tag{2.6}$$

The representation of a general or partially specified signal requires a mechanism for representing signals whose sample values and properties are not completely known.

Comparison of Computational Cost

To provide a high-level compiler of signal processing algorithms, the environment must include some method for ranking the mathematically equivalent implementations of a given algorithm and selecting the best one. The metric that is generally used by compilers is the computational cost of the alternative algorithms. Thus, the signal processing environment must be able to determine the relative costs of equivalent algorithms, in order to select the most computationally efficient. Myers [2] and Covell [1] discuss computational cost metrics in more detail.

2.3.5 Summary of Requirements for an Integrated Signal Processing Environment

An integrated signal processing environment should support the numeric processing of signals, the definition of new signal processing algorithms, the analysis of the properties of signals and systems, and the rearrangement or "compilation" of signal processing algorithms. To provide support for these capabilities, certain constraints have been placed on the signal processing environment.

- It must provide an explicit representation for signals.
- The sample values of signals must be accessible at random from anywhere in

the domain of the signal: this requirement implies both that the external access of the sample values is unaffected by the internal computational model and that sample values must be accessible outside the nonzero support of signals.

- The signal must be immutable in its external characteristics: its property values and sample values must be unchanging from the time they are first referenced.
- Analyses of signal and system properties and transforms must be provided by the environment, using information included in the definitions of signal and system classes.
- Finally, algorithm rearrangement and cost analysis must also be provided by the computer, to allow for high-level compilation of signal processing algorithms.

Figure 2.4 reviews the capabilities of some currently available signal processing environments. This figure does not attempt to exhaustively list the currently available software. Instead an attempt is made to examine the range of signal processing environments currently in use. The first set of signal processing environments [17, 18] listed in Figure 2.4 were developed to only support the definition and numerical application of completely specified, numeric algorithms. As mentioned in section 2.1 and described in Chapter 1, SRL is the result of research by Kopec into data abstractions both to reflect the basic characteristics of signals and to support numeric manipulations. Kopec [3] advocated the immutability of signals and the explicit availability of their nonzero supports as being essential for simplifying and clarifying signal processing programs. nthPOWER (Signal Technology, Inc.) is a commercial version of SRL and one of its descendants, ISPUD [8]. SPLICE [2, 4] was also mentioned in section 2.1. In SPLICE, as in SRL, sequences are immutable data objects with an explicit nonzero support. Unlike SRL, sample values outside the nonzero support can be accessed by the same operations that access the sample values inside the nonzero support. Sequences, defined by the generating system and its inputs, behave uniformly independent of the signal processing model used to define the system: for example, sample values of a sequence defined using a state-machine model can be fetched at any index without explicitly determining the

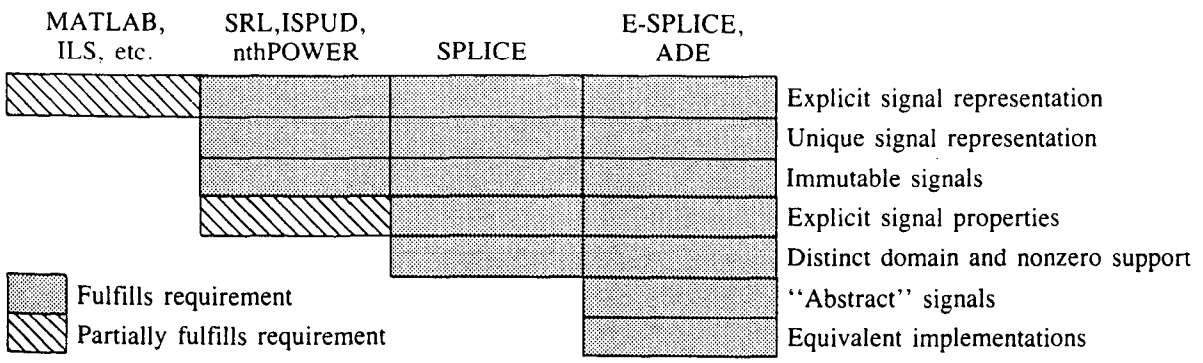


Figure 2.4 The capabilities of some currently available signal processing environments.

previous states. E-SPLICE [2] and its descendent, ADE [1], were designed to support not only numerical processing and algorithm definition but also automated property analysis and algorithm rearrangement. Hence, these are only environments that provide signal representations which meet most of the constraints developed in this section. In particular, the signal representations are unique and immutable with a distinct domain and nonzero support and with explicit signal properties, such as symmetry and computational cost. Furthermore, these environments automate the analysis of properties and signal transforms and can represent and manipulate "general" signals. Finally, both E-SPLICE and ADE provide a tool that approaches the desired high-level signal processing compiler discussed earlier: both environments will provide an enumeration of alternate implementations of an algorithm, partially ranked on the basis of the computational cost vectors.

The remainder of this chapter focuses on the capabilities and characteristics of ADE. Section 2.4 includes an example of the use of ADE to define and analyze the properties of the FSK-code detector, shown in Figure 2.3. The remaining sections of the chapter explore the process through which alternate implementations of an algorithm are found by ADE. Some results from using ADE to find alternate implementations are discussed in section 2.4.

2.4 ADE: A FEASIBILITY PROOF FOR AN INTEGRATED SIGNAL PROCESSING ENVIRONMENT

The previous section pointed out some of the requirements for a complete and well-integrated signal processing environment. As shown in Figure 2.4, ADE attempts to meet each of these requirements. Therefore, ADE will be used within this chapter to illustrate the potential benefits of a well-integrated signal processing environment. The illustration of these potential benefits is simplified by considering a specific signal processing design problem: the problem that has been chosen for this illustration is the design of a detector for sonar FSK-code reflections.

This section provides a brief description of ADE. It is guided by a discussion of two short sessions in ADE, one illustrating the programming of the environment and the other, its interactive use.

2.4.1 An Overview of ADE

ADE provides an integrated environment for numerical processing, algorithm definition, signal and system property analysis, and algorithm rearrangement. The underlying signal representation is an object-oriented signal representation, satisfying all the desirable properties discussed in section 2.3. Information about signal processing, used in property analysis and in algorithm rearrangement, is represented

in a rule-based system. Although the capacity for forward chaining[5] is included in ADE, the majority of the environments resources are devoted to backward chaining from specific inquiries. Tools are provided within ADE for extending this rule base, both through the introduction of new signal and system classes and through the introduction of new properties and signal transforms.

ADE is a descendant of the SPLICE and E-SPLICE environments. ADE inherits its basic approach to signal definition and representation from SPLICE. The influences of E-SPLICE and to a lesser extent PDA [9] are reflected in the structure of some parts of the rule base. In particular, as in E-SPLICE, ADE uses backward-chaining rules to describe the properties of signals. ADE, like E-SPLICE, supports multilevel matching within the patterns of these rules. The approach used in ADE for matching forward-chaining rules was introduced by Dove [9]. ADE makes use of a subset of QM [10] and a limited number of functions from MACSYMA [11]. QM is the product of research into qualitative mathematics. It represents, manipulates, and describes piecewise-continuous functions. A subset of QM is used to record and propagate constraints on symbolic numbers. ADE includes an extension to QM to support limited reasoning about symbolic integers as well as the continuously variable numbers. ADE also makes limited use of MACSYMA to simplify and factor the polynomials used in the characterization of z-transform signals. ADE is written in Symbolics Common Lisp [12]. This choice of language provides both the flexibility of a LISP dialect and support for object-oriented programming.

The remainder of this section provides examples of the use of ADE in the context of the FSK-code problem introduced earlier. Examples are given of programming (algorithm definition), property analysis, and algorithm rearrangement in ADE.

2.4.2 Examples of Algorithm Definition and Manipulation in ADE

In the sonar imaging problem, an important problem is to find a way to achieve good spatial resolution without requiring a large, costly array and without missing transients through the use of a swept beam. The first step in solving this problem is to select a method by which it will be solved. Although this selection draws heavily on signal processing experience and creativity, the selection process can be accelerated by providing a support environment in which the signal and system representations closely match the internal models used by the system designer. These representations must change according to the problem at hand, since different problems give

[5] Forward chaining in a blackboard/rule-based system is the use of current information to determine additional information, without requiring an inquiry to explicitly trigger that line of reasoning. In general, this approach triggers any rule whose preconditions are satisfied based on the current state of the blackboard. This contrasts with backward chaining, which starts from a question and then, based on the preconditions of rules that could answer that question, asks additional questions until a question is asked that can be answered.

rise to different signal models. To provide this range of representations, ADE allows the system designer to introduce his own signal and system definitions. For example, in the FSK-code detector shown in Figure 2.3, the incoherent combination of the matched filter outputs is modeled as a single processing block which follows, but is separate from, the matched filtering itself. To support this model of the detector, a new system class, INCOHERENT-COMBINATION, is defined in Figure 2.5. The definition relies on the composition of other, previously defined signal processing systems to provide the output signals with their observable characteristics: lines 8–12 of Figure 2.5 describe this composition.

To simplify the programming task, signal and system definitions closely mimic the notational conventions used in signal processing. As illustrated by lines 1–4 of Figure 2.5, signal and system definitions form new "classes" of signals and systems. Hierarchies of classes are used to make similarities explicit and to reduce the amount of coding required. Signals are formed by one of two paths: either as independent entities that are inherently defined, like an impulse or a complex-exponential sequence, or as the output from a system that has been applied to some inputs. Some of the 43 inherent signal classes and the 169 system classes currently defined in ADE are listed in Table 2.2.

Once all the appropriate signal and system classes have been defined, the process of creating and analyzing the signals and systems involved in the design problem is simplified. Figure 2.6 illustrates the design sequence. Line *I-1* of Figure 2.6 shows the definition of a partially specified discrete-time signal, $x[n]$, which is periodic and complex valued with both its real and imaginary parts in the range from -1 to $+1$. This description is only a partial description of the input since there are a large number of discrete-time sequences that satisfy all parts of this description. The resulting object, printed on line *O-1*, is an "abstract" signal.

The FSK-code detector is compactly described in lines *I-2* and *I-3* of Figure 2.6. As can be seen by comparing these lines with the model for the detector shown in Figure 2.3, the computer representation and the designer's representation are closely matched. As is shown in the remainder of this figure, ADE provides information about the properties of the output signals from this detector and about alternate implementations of the matched filters used in the detector.

In more detail, lines *I-2* and *I-3* of Figure 2.6 create an incoherent detector for the set of 16 FSK codes, using a 16-point rectangular window to shape each frequency chip. The input to this detector is $x[n]$. The output from the modulated filter bank is a two-dimensional signal, YC. YC is used as the input to the incoherent combiner and the final output signal is Y. Lines *I-4* and *I-5* of Figure 2.6 request the range and periodicity of the output from the detector, Y. The range and periodicity of a signal are among the properties that can be explicitly requested. Signal properties, such as symmetry, sample type, and nonzero support, are explicitly available characteristics of every signal. Similarly, system properties, such as equivalent systems and invertibility, are explicitly available characteristics of every system. ADE determines the values of signal and system properties by using the property information explic-

ADE Code	Description
1 (DEFINE-SYSTEM-CLASS-ALIAS	1
2 (INCOHERENT-COMBINATION N@INTEGER)	2 accept an integer (N) as a system parameter
3 (INPUT@2D-SEQUENCE)	3 accept a 2-D sequence (INPUT) as an input
4 (SHIFT-INVARIANT-SYSTEM HOMOGENEOUS-SYSTEM 2D-SYSTEM)	4 a subclass of these classes
5 ("an incoherent combiner of shifted versions of the sequences in INPUT"	5
6 SELF)	6 the systems "alias" to themselves
7 ('the output sequences from the incoherent combination'	7 the output signals "alias" to the composition of operations:
8 (MAP-OVER BANK-OF-SEQUENCES I 0 N	8 (BANK-OF-SEQUENCES $S_0 \ldots S_{N-1}$)
9 (MAP-OVER SEQUENCE-ADD K 0 N	9 where S_i = (SEQUENCE-ADD $p_i(0) \ldots p_i(N-1)$)
10 (OUTPUT-OF(SEQUENCE-SHIFT (* K N))	10 where $p_i(k)$ = (OUTPUT-OF (SHIFT (* k N))
11 (SEQUENCE-MAGNITUDE	11 (MAGNITUDE
12 (FETCH-SEQUENCE INPUT (MOD(− (*(1 + I)(1 + K))1)(1 + N)))))))))	12 $\text{INPUT}_{((i+1)*(k+1)-1) \bmod (N+1)}$))

Figure 2.5 An example of the programming of ADE
This example defines an incoherent combiner for the FSK system of Figure 2.3 in the special case when the permutation function, $p_i(k) = ((i + 1)*(k + 1) - 1) \bmod (N + 1)$

TABLE 2.2 SOME OF THE SIGNAL AND SYSTEM CLASSES CURRENTLY DEFINED IN ADE

Inherent signal classes (43 hierarchical classes)	
DISCRETE-TIME-SEQUENCE	COMPLEX-EXPONENTIAL
FOURIER-DOMAIN-SIGNAL	CAUSAL-RECTANGULAR-WINDOW
Z-DOMAIN-SIGNAL	SINC
2D-SEQUENCE	COSINE-SEQUENCE
RATIONAL-ZT	SINE-SEQUENCE
CONSTANT	FIR-SEQUENCE
POWER-SEQUENCE	IIR-SEQUENCE
IMPULSE	CAUSAL-IIR-SEQUENCE
GENERAL-EXPONENTIAL	ANTICAUSAL-IIR-SEQUENCE
UNIT-STEP-SEQUENCE	STABLE-IIR-SEQUENCE
CAUSAL-HAMMING-WINDOW-SEQUENCE	

System classes (169 hierarchical classes)		
DISCRETE-TIME-SYSTEM	ADD	BANK-OF-SEQUENCES
2D-SYSTEM	SUBTRACT	ROTATED-BANK-OF-SEQUENCES
FOURIER-DOMAIN-SYSTEM	MULTIPLY	SHORT-TIME-WINDOW
Z-DOMAIN-SYSTEM	CONVOLVE	SHORT-TIME-FT
SHIFT-INVARIANT-SYSTEM	SHIFT	MAPOVER-SYSTEM
GENERALIZED-SHIFT-INVARIANT-SYSTEM	SCALE	FIR-FILTER
MEMORYLESS-SYSTEM	RECIPROCAL	CAUSAL-IIR-FILTER
ASSOCIATIVE-SYSTEM	DIVIDE	ANTICAUSAL-IIR-FILTER
ADDITIVE-SYSTEM	REAL-PART	SIGNAL-ALIAS-IN-2PI
HOMOGENEOUS-SYSTEM	IMAG-PART	FOURIER-TRANSFORM
GENERALIZED-HOMOGENEOUS-SYSTEM	MAGNITUDE	INVERSE-FOURIER-TRANSFORM
LINEAR-SYSTEM	INPUT-PHASE	Z-TRANSFORM
GENERALIZED-LINEAR-SYSTEM	ABSOLUTE-VALUE	INVERSE-Z-TRANSFORM
SEQUENCE-CIRCULAR-SHIFT	SCALE-INDEX	INVERSE-TRANSFORM
SEQUENCE-CIRCULAR-REVERSE	UPSAMPLE	DISCRETE-FOURIER-TRANSFORM
SEQUCNCE-CIRCULAR-CONVOLVE	DOWNSAMPLE	COMPLEX-CONJUGATE
SEQUENCE-CONVOLVE-OVERLAP-SAVE	INTERLEAVE	

Interactive sessions	Description
I-1 ADE: (NAMED-SETQ X (A-MEMBER-OF 'DISCRETE-TIME-SEQUENCE &PROPERTIES :PERIODICITY 256 :RANGE (CREATE-RANGE {−1 1} {−1 1}))))	I-1 ADE: create X, an "abstract" discrete-time sequence with a periodicity of 256 samples and with real and imaginary sample values above −1 and below 1
O-1 ⇒ X	O-1 ⇒ the abstract discrete-time sequence
I-2 ADE: (NAMED-SETQ YC (OUTPUT-OF (MODULATED-FILTER-BANK (REVERSE (RECTANGULAR-WINDOW 16)) 16) X))	I-2 ADE: create YC, the output of a 16-channel modulated filter bank with a 16-point anticausal rectangular window as the base impulse response and with X as the input to the filter bank
O-2 ⇒ YC	O-2 ⇒ the output from the modulated filter bank
I-3 ADE: (NAMED-SETQ Y (OUTPUT-OF (INCOHERENT-COMBINATION 16) YC))	I-3 ADE: create Y, the output of the incoherent combiner with YC as the input
O-3 ⇒ Y	O-3 ⇒ the output from the incoherent combiner
I-4 ADE: (RANGE Y)	I-4 ADE: what is the range of Y?
O-4 ⇒ (RANGE {0 32} {0 0})	O-4 ⇒ Y is real with values between 0 and 32
I-5 ADE: (PERIODICITY Y)	I-5 ADE: what is the periodicity of Y?
O-5 ⇒ 256	O-5 ⇒ the periodicity of Y is 256 samples
I-6 ADE: (EFFICIENT-IMPLEMENTATIONS YC)	I-6 ADE: what are the efficient ways to compute?
O-6 ⇒ ((MAP-OVER 'BANK-OF-SEQUENCES K 0 16 (MAP-OVER (IF (< K 8) 'ADD 'SUBTRACT) N8 0 2 (OUTPUT-OF (SCALE (EXP (COMPLEX 0 (* −2 PI K N8)))) . . . .))) (BANK-OF-SEQUENCES (SEQUENCE-ADD . . .) . . .)	O-6 ⇒ the list of efficient implementations includes the classic FFT implementation pruned FFT implementation shown in Figure 2.7 and other, partially pruned FFT implementations

Figure 2.6 A sample of an interactive session in ADE
The I-lines with the token "ADE:" designate the user's inputs and the O-lines with the token "⇒" designate the outputs. See Covell (1989) for a more detailed discussion of the use of ADE.

itly included within the definitions of signal and system classes or, if this information is missing, by using the default value for the property. Some of the signal and system properties that are currently included in ADE were listed in Table 2.1. Tools are also available within ADE for adding other signal properties that could be useful to the particular problem under consideration (e.g., stationarity).

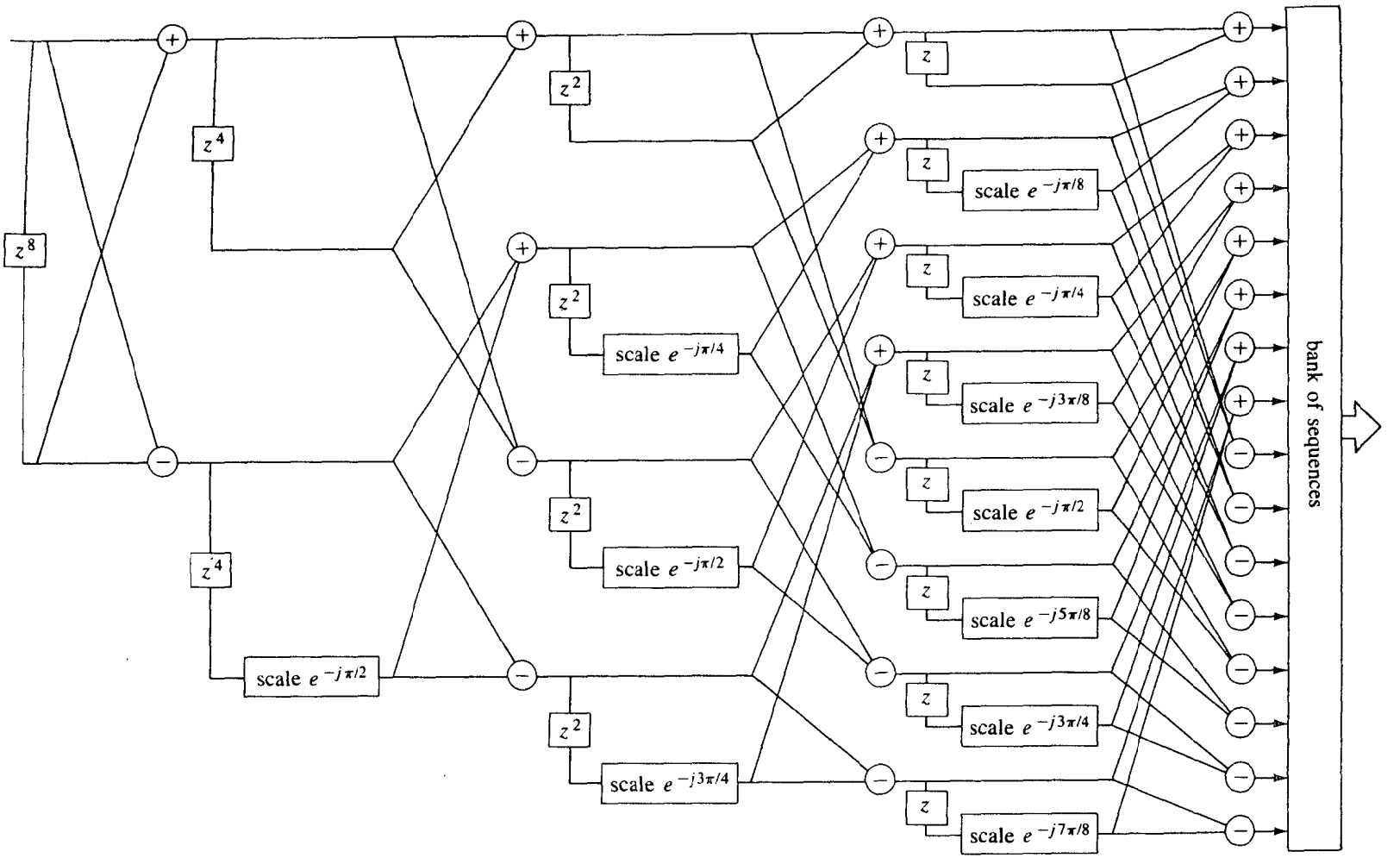
Line *I-6* of Figure 2.6 requests a list of all the computationally efficient implementations that can be found for the matched filters, used in the detector. Computational efficiency is determined in ADE on the basis of memory requirements and the required additions, multiplications, and memory references. Line *O-6* shows some of the efficient implementations generated by ADE. The given form of the modulated filter bank is not included in this list of implementations, since there are other implementations that are more computationally efficient. Instead, the list includes the classic FFT-based structure, described in section 2.2, and a variety of "pruned" FFT-based structures: one of the pruned FFT-based structures is shown in Figure 2.7.[6] The pruned FFT implementations have the same underlying structure as the classic FFT implementation. The difference lies in the *number* of butterflies that are computed at each stage. For example, the pruned FFT structure shown in Figure 2.7 has only one butterfly in the first stage, two in the second, four in the third, eight in the fourth, and so on, while the classic FFT structure has $N/2$ butterflies in each stage. As can be seen from the comparison of costs shown in Figure 2.7, a trade-off exists between the minimum number of memory locations, achieved by the classic FFT structure, and the minimum number of operation counts, achieved by the pruned FFT structure.[7]

Since the selection of the frequency-chip window affects both the range resolution and the signal-to-signal rejection of the sonar system, another two frequency-chip windows were considered to explore the trade-offs between resolution and signal-to-signal rejection: the 16-point Hanning window and the 32-point Hanning window with overlapping frequency chips within the FSK code.[8] Figure 2.8 shows the pruned FFT structure generated by ADE for the 16-point Hanning window. The resulting structure consists of the pruned FFT structure of Figure 2.7 followed by frequency-domain convolution. Figure 2.9(a) shows the pruned FFT structure gen-

[6] Figure 2.7 shows operations of the form z^i for positive values of i. These operations are (anticausal) sequence advance operations and arise from the use of the anticausal window within the modulated filter bank (see line *I-2* of Figure 2.6). The use of the anticausal window within the modulated filter bank is the result of using a causal window for the frequency chips in the FSK codes [see (2.1)].

[7] The memory counts do not include the registers necessary for storing the intermediate sequence values. If these additional memory locations were included in the cost structures, the amount of memory for the classic FFT structure using the method given by Singleton [13] and the pruned FFT structure would be identical.

[8] These requests for efficient implementations of the two Hanning-window modulated filter banks are not shown in Figure 2.6 since they are not substantially different in form from the request already shown in that figure.



Structure	Computational costs for $N = 16$: Complex multiplies	Complex adds	Shift registers	Approximate computational costs for general N: Complex multiplies	Complex adds	Shift registers
Modulated filter bank	225	256	15	N^2	N^2	N
General FFT structure	17	64	15	$\frac{N}{2}\log_2\frac{N}{2}$	$N\log_2 N$	N
Pruned FFT structure	11	30	24	N	$2N$	$\frac{N}{2}\log_2 N$

Figure 2.7 The pruned FFT implementation of the modulated filter bank for a 16-point rectangular window.

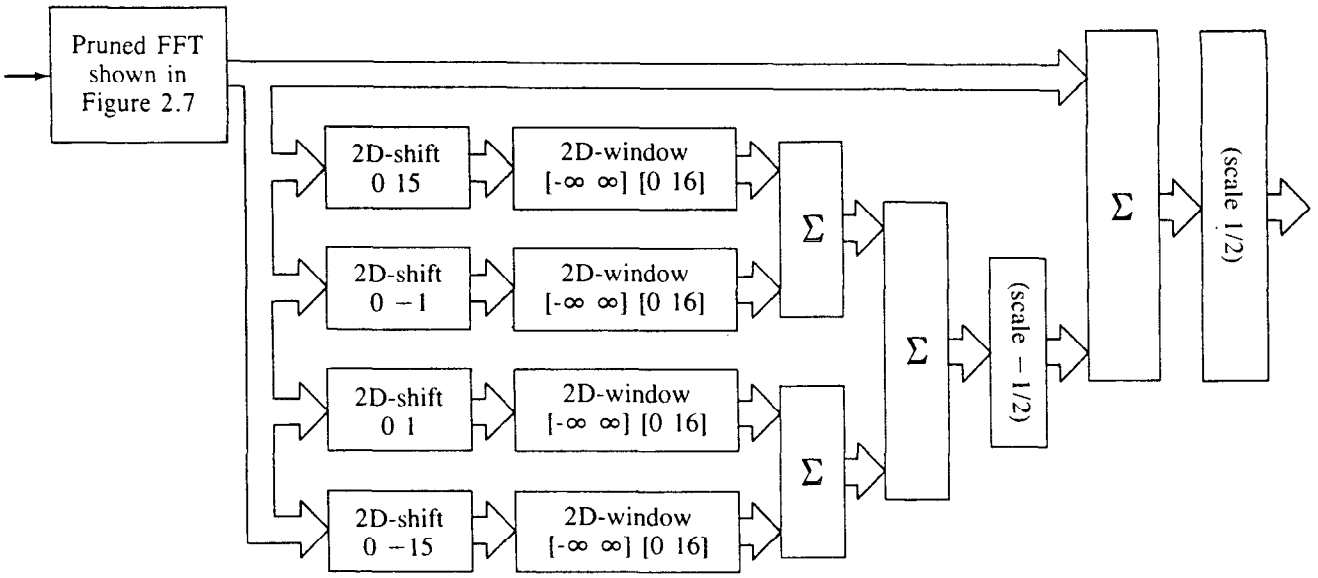
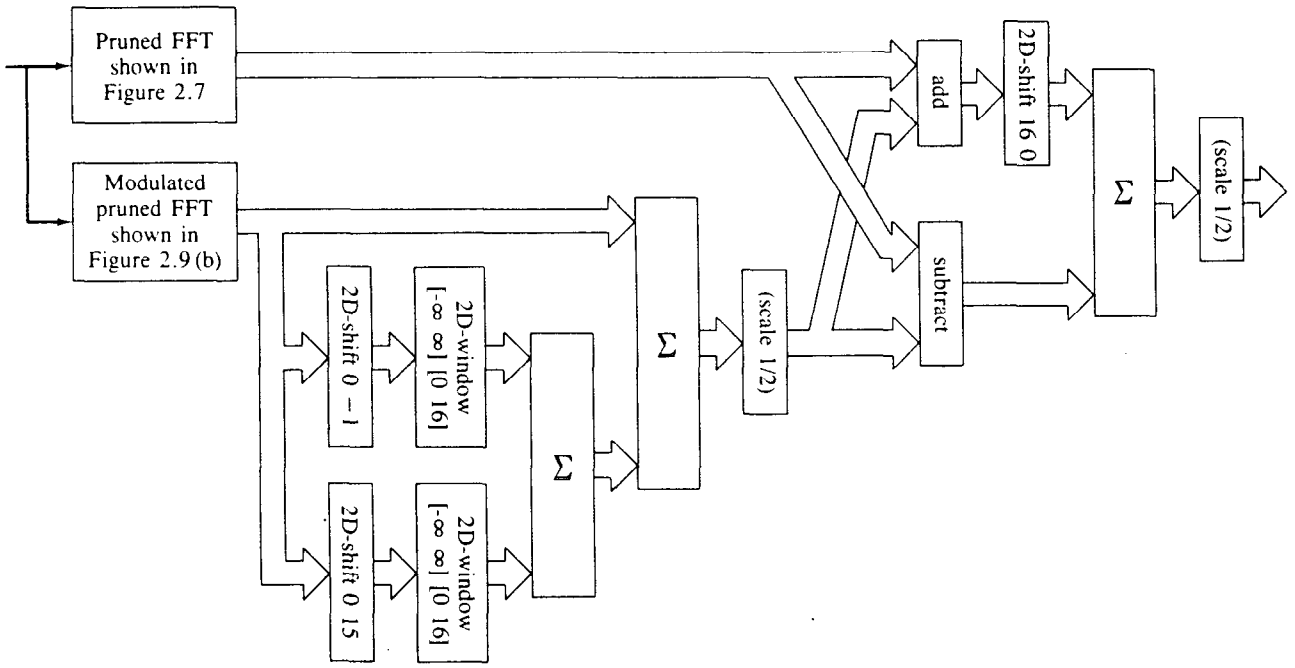
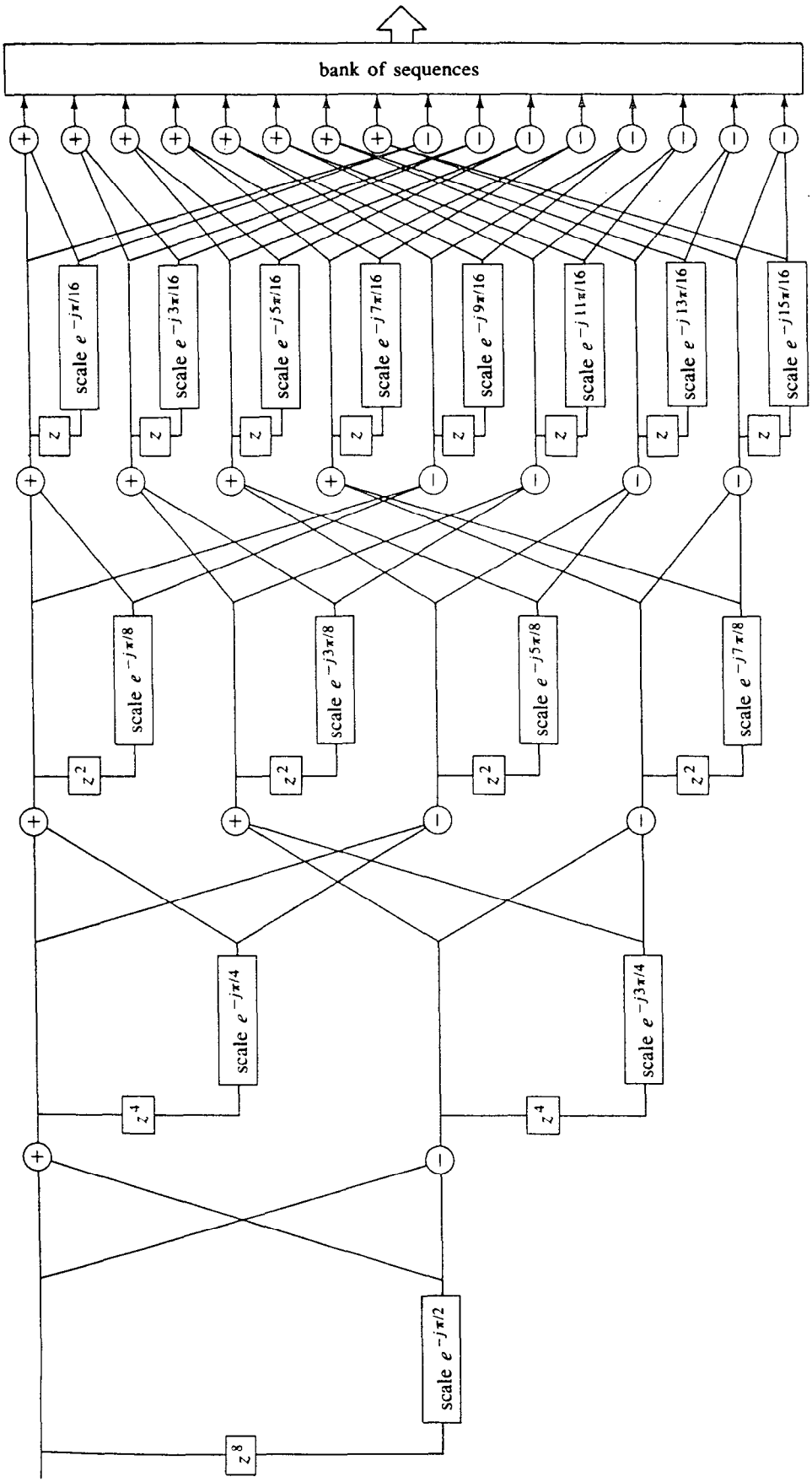


Figure 2.8 The pruned FFT implementation of the modulated filter bank for a 16-point Hanning window.



(a)

Figure 2.9 The pruned FFT implementation of the 16-channel modulated filter bank using a 32-point Hanning window.



(b) The modulated pruned FFT structure.

erated by ADE for the overlapping, 32-point Hanning window. The resulting structure uses two pruned FFTs, one taken from Figure 2.7 and the other illustrated in Figure 2.9(b), followed by frequency-domain convolution. The modulated pruned FFT structure of Figure 2.9(b) computes a 16-point pruned FFT, offset by half a frequency bin. As with the rectangularly windowed design, the implementations that were found reduced the computational complexity of the matched filter bank to $O(N)$.

It is interesting to note that, with the pruned FFT, the *order* of the computational complexity is actually reduced as well as the number of computations themselves. Specifically, the order is reduced from $O(N^2)$ for the direct-form implementation or from $O(N \log N)$ for the classic FFT implementation to $O(N)$ for the pruned FFT implementation. The amount of computation that is required for the pruned FFT is actually identical to that of the recursive implementation and, as mentioned earlier, the pruned FFT structure has the advantage of being numerically stable while the recursive formulation is marginally unstable due to its reliance on pole/zero cancellation on the unit circle.

Another interesting aside is that the pruned FFT structure has not been found in the currently published literature. Although other pruned FFT structures have been published [14, 15], these structures depend on the characteristics of the inputs as opposed to the characteristics of the desired outputs. Thus, in addition to the standard implementations, ADE generated a new structure for efficiently computing the outputs from a modulated filter bank.

ADE generates equivalent implementations of an algorithm by testing and applying algorithm transformation rules. These rules are included within the definitions of signal and system classes. Detailed derivations of the pruned FFT implementations of the modulated filter banks are provided in Appendix A. In the derivation of the pruned FFT implementation of the rectangularly windowed modulated filter bank, the actual transformation rule, which is crucial, is relatively straightforward. The crucial transformation rule simply pulls common shifts through a generalized shift-invariant system. With a generalized shift-invariant system, $H\{\ \}$, if $y(t) = H\{x_1(t), \ldots, x_N(t)\}$ then $y(t - T) = H\{x_1(t - T), \ldots, x_N(t - T)\}$. Lines 13–30 of Figure 2.10 show how the shift-invariant property of the generalized shift-invariant system is used in ADE to generate an equivalent form. By pulling all the common shift operations through the butterfly and twiddle stages of the classic FFT structure, the classic FFT structure collapses into the pruned FFT structure. The derivations of the pruned FFT implementations of the other two windows are longer and more involved but, again, rely only on relatively straightforward transformations.

As illustrated with the rule describing the shift-invariant property of the generalized shift-invariant system, the transformation rules that generate the equivalent implementations are relatively straightforward. However, as illustrated in section 2.3.4, the search process that must be used to discover and combine all the appropriate transformations is comparatively complex. The remaining sections of this chapter focus on this search process.

ADE code	Description
(DEFINE-ABSTRACT-SYSTEM-CLASS	
(GENERALIZED-SHIFT-INVARIANT-SYSTEM *)*	accept any parameters or inputs
( )	no superclasses
("h{ } s.t. y[n] = h{x_1[n] . . . x_L[n] ⇒ y[n − N] = h{x_1[n − N] . . . x_L[n-N]}")	
nil ( )	generate a new output signal class
("the output from a generalized shift-invariant system"	
(GOAL SIMPLIFICATION	a simplification rule: if all the inputs are shifted identically,
:NAME SHIFTED-INPUT	pull shift system outside.
:OBJECT (OUTPUT-OF ?SYSTEM@(NOT SHIFT-SYSTEM) &REST	the inputs are all outputs from a single shift system.
?INPUTS${(OUTPUT-OF ?SHIFT ?[SHIFT-INPUTS])})	?SHIFT-INPUTS will be bound to the list of shift inputs
:ANSWER (OUTPUT-OF SHIFT	
(APPLY 'OUTPUT-OF SYSTEM SHIFT-INPUTS)))	
(GOAL EQUIVALENT-FORM	an equivalent form rule: if all the inputs are shifted,
:NAME UNEQUALLY-SHIFTED-INPUT	pull one of the shifts outside.
:OBJECT	
(OUTPUT-OF ?SYSTEM@(NOT SHIFT-SYSTEM) &REST	the inputs are all outputs from shift systems.
?INPUTS${(OUTPUT-OF (SPECIFIC-MEMBER SHIFT-SYSTEM	?SHIFT-FACTORS will be bound to the list of shift amounts.
&REST ?[SHIFT-FACTORS])	?SHIFT-INPUTS will be bound to the list of shift inputs.
?[SHIFT-INPUTS])})	
:ANSWER	
(LET ((COMMON-SHIFT (FIRST SHIFT-FACTORS)))	pull first shift outside
(OUTPUT-OF (APPLY 'SHIFT COMMON-SHIFT)	
(APPLY 'OUTPUT-OF SYSTEM	
(MAPCAR	
'(LAMBDA (SHIFT-FACTOR SHIFT-INPUT)	
(LET ((REMAINING-SHIFT	compensate for the shift that was pulled outside
(MAPCAR '$- SHIFT-FACTOR COMMON-SHIFT)))	
(OUTPUT-OF (APPLY 'SHIFT REMAINING-SHIFT)	
SHIFT-INPUT)))	
SHIFT-FACTORS SHIFT-INPUTS)))))	
. . .))	additional information about generalized shift-invariant system outputs

Figure 2.10 The definition for the system class, GENERALIZED-SHIFT-INVARIANT-SYSTEM

2.5 UNCONSTRAINED DERIVATION AND RANKING OF EQUIVALENT ALGORITHMS

To simplify the development of efficient algorithms, an integrated signal processing environment should provide a high-level, signal processing "compiler." This compiler must make use of rules, such as the one shown in Figure 2.10, to explore the space of alternative implementations. This section and the next describe the structure of the search space that is explored in finding the equivalent implementations of an algorithm.

2.5.1 Unconstrained Search for Equivalent Algorithms

The task of finding alternate implementations of a signal processing expression is the same as finding all the algorithmic transformations that are applicable to the signal processing expression; to its input/output equivalent expressions; and to the subexpressions used by these expressions. For example, to find the equivalent implementations of the filter bank used in the FSK-code detector, all the applicable algorithmic transformations for the filter bank should be completed, as should the transformations on the modulated window sequences and the input sequence. In addition, once an alternate implementation is generated, all of the algorithmic transformations that are applicable to this new expression or to one of its subexpressions must also be applied. Thus, equivalent implementations of a signal processing expression can be obtained in any of a variety of ways: a transformation can be applied to the original signal processing expression itself; a subexpression of the original expression can be replaced by an equivalent implementation of the subexpression; or either of these approaches can be applied to one of the newly generated equivalent implementations of the signal processing expression.

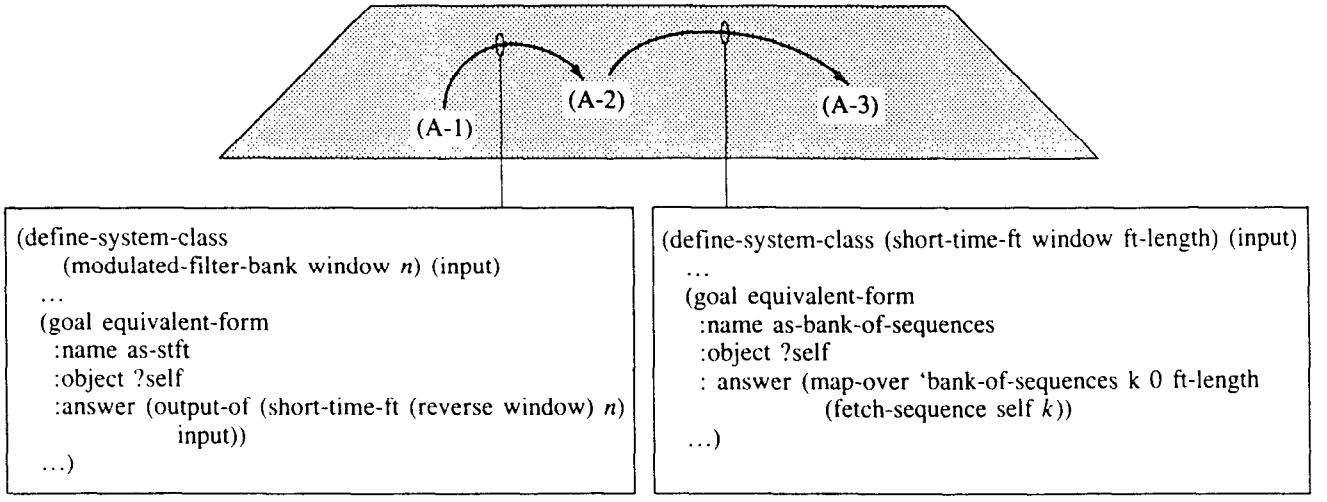
To simplify this discussion, a graphical representation of the search process is presented in Figure 2.11. The problem of finding the equivalent forms of a signal processing expression, without consideration of its subexpressions, can be represented graphically as a planar net, as shown in Figure 2.11(a). The nodes of the net represent the signal processing expression and its equivalent forms. The directed arcs connecting the nodes within the planar net represent the application of simple transformation rules. For example, in Figure 2.11(a), the modulated filter bank (node A-1) is replaced by a short-time Fourier transform (node A-2) using a rule included in the definition of the system class MODULATED-FILTER-BANK: this rule is shown below the transformation arc.

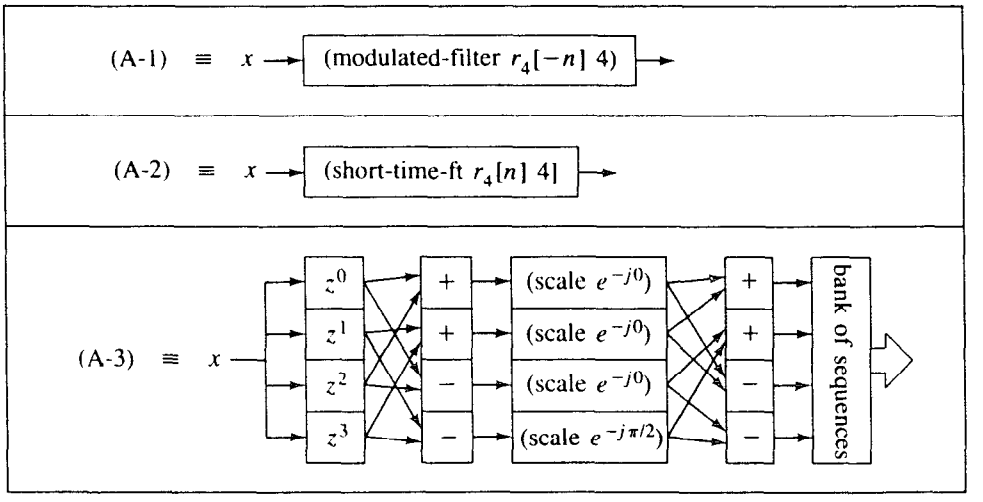
The new nodes that result from algorithm transformations can themselves be used as the starting point for other transformations. An example of this recursive transformation is also shown in Figure 2.11(a): the modulated filter bank is first replaced by a short-time Fourier transform, which is then expanded into the basic addition, subtraction, time shifts, and multiplications (node A-3) that make up the short-time Fourier transform.

Each of the nodes of the planar net can also be viewed as a combination of

subexpressions: the subexpressions are the inputs to the generating system. For example, as shown in Figure 2.11(b), the four inputs to the BANK-OF-SEQUENCES system in node A-3 can each be manipulated independently. In particular, each of these four expressions can be replaced by any of their equivalent implementations, without changing the input/output mapping of the overall algorithm. Thus, this replacement provides additional equivalent implementation to the original signal processing expression.

Graphically, requesting the equivalent forms of the inputs to the generating system of a signal drops the problem down to another set of nets and again tries to

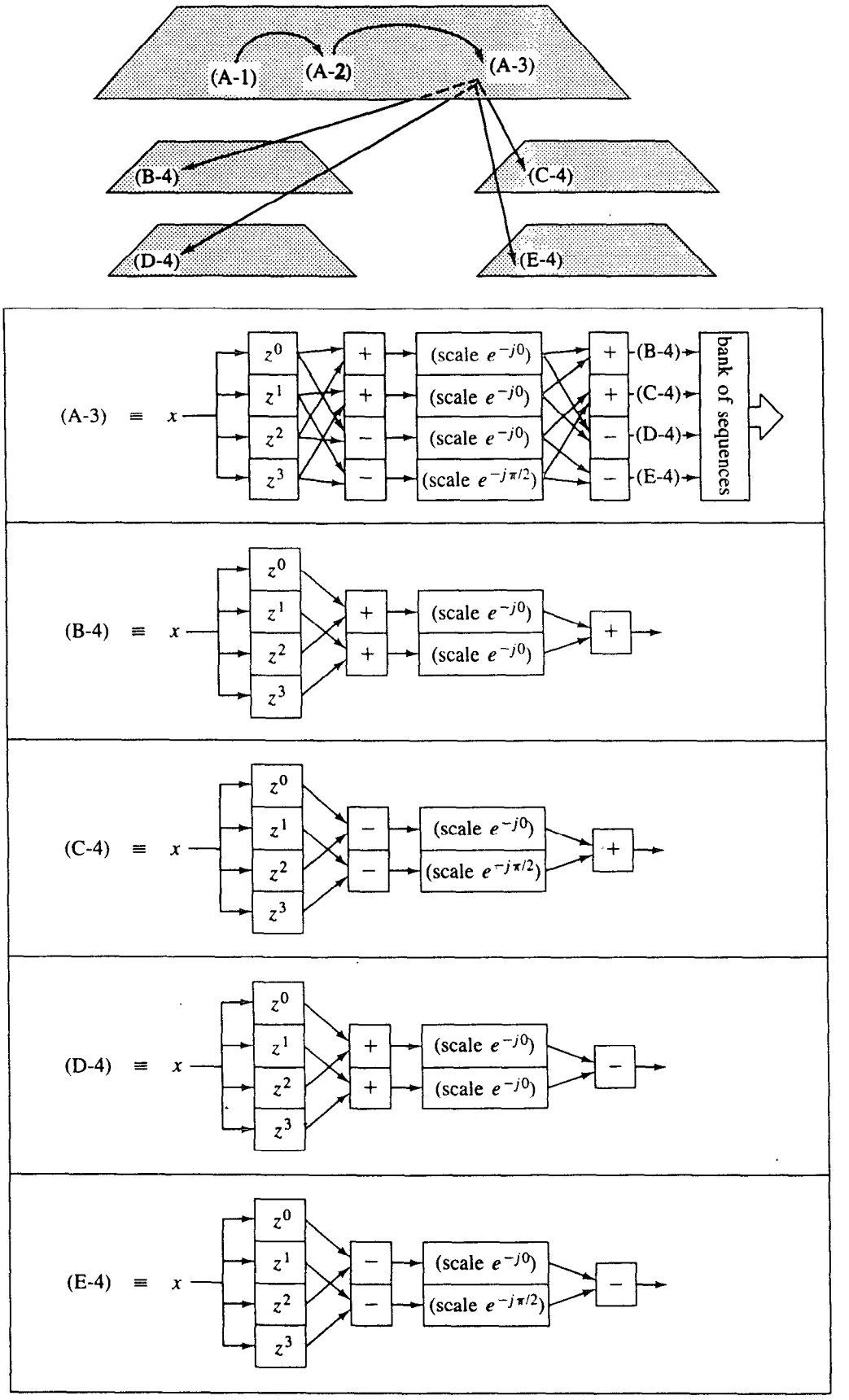




$r_4[n]$ = (rectangular-window 4)

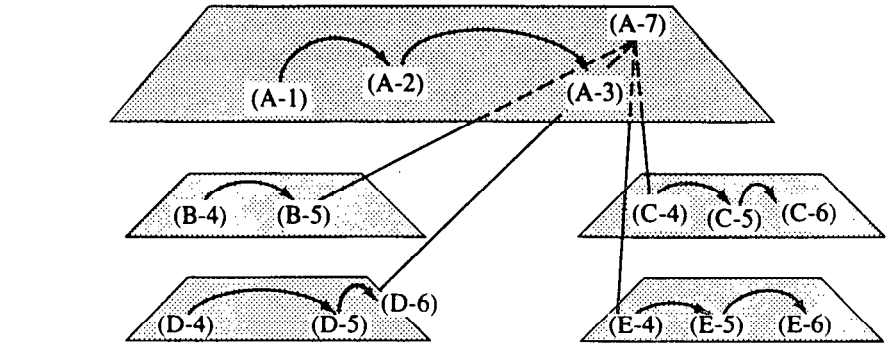
(a)

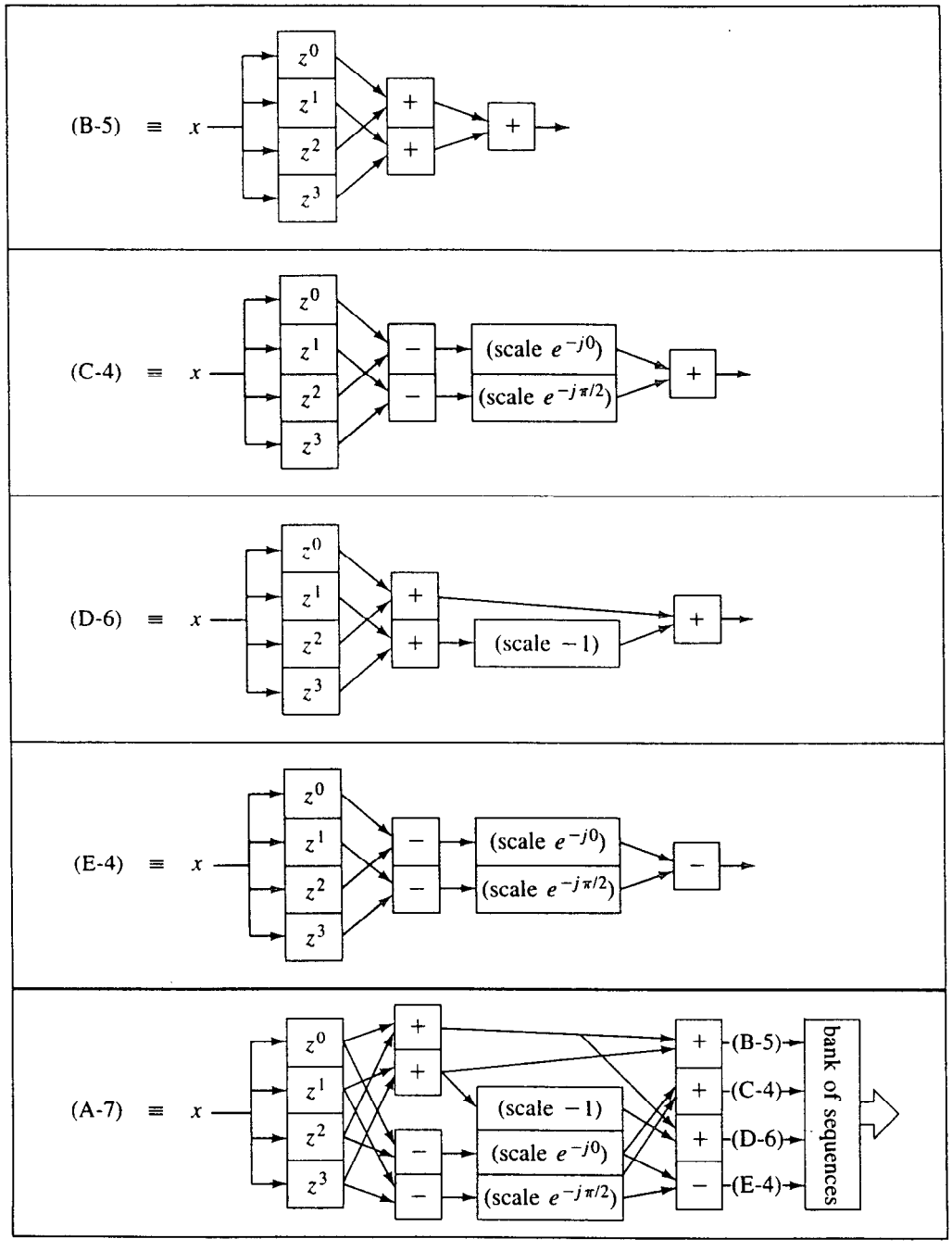
Figure 2.11 A net representation of the search for equivalent forms. This figure shows an example of a search for equivalent forms. Each node (e.g., A-1 or D-4) represents an expression. The name for each node consists of a letter (A through E) and a number. The letter indicates which expressions are equivalent (e.g., D-4 is equivalent to D-5) and the number indicates the order within the sequence of manipulations (e.g., D-5 is modified to create D-6).



(b)

Figure 2.11 (continued)





(c)

Figure 2.11 (continued)

find connected nodes. In our example, this process generates four new subsearches for the equivalent forms of the four inputs to the BANK-OF-SEQUENCES system in node A-3: Nodes B-4, C-4, D-4, and E-4 represent those inputs in Figure 2.11(b). The subsearches must also find all equivalent implementations of their given algorithm using simple transformations, repeated transformations, and subexpression transformations.

Once all the equivalent forms of the inputs are found, these equivalent forms can be used to replace the original input expressions. This replacement process is shown in Figure 2.11(c) as a projection of the nodes on the lower planar nets back into the original net: nodes B-5, C-4, D-6, and E-4 are used as inputs into the BANK-OF-SEQUENCES system, resulting in the new expression, node A-7. As with this example, the projection upward often generates new nodes in the original net (node A-7). A new node in the original net is generated whenever the input replacement results in an expression that has not already been generated through some other transformation path. These new nodes are equivalent forms of the original expression: thus, the new nodes can also be the starting point for further transformations.

This process of repeated transformation and subexpression transformation continues until no new equivalent expressions (nodes) can be found.

2.5.2 Infinite Expansion to the Search Space for Equivalent Algorithms

Two major difficulties with the search process described above are apparent after careful consideration: the possibility of the infinite expansion of the search space and the finite but exponential growth of the space due to the separate manipulation of subexpressions. The search space will expand indefinitely, if a simple transformation or a combination of simple transformations repeatedly introduce operators that have no net effect (e.g., a delay operator followed by an advance operator). This difficulty is considered briefly in this subsection. The problem of limiting the exponential growth is the subject of the next section of this chapter.

The transformations used in generating equivalent implementations often result in signal processing algorithms whose complexity is greater than the manipulated algorithm. For example, consider the problem of finding equivalent implementations of the matched filters for the frequency chips in the rectangularly windowed FSK-code detector. One of the rearrangements that is found is shown in Figure 2.12(a). The subexpressions of this implementation will be manipulated, due to the combination of recursive search and subexpression manipulation. One of the implementations discovered by this process is shown in Figure 2.12(b): this structure results from the application of a rule shown on lines 13–30 of Figure 2.10. Applying repeated transformation and subexpression manipulation to the structure shown in Figure 2.12(b) will result in, among others, the structure shown in Figure 2.12(c). In fact, the recursive transformations and the increasing complexity of the algorithmic description would result in the infinite expansion of the search space.

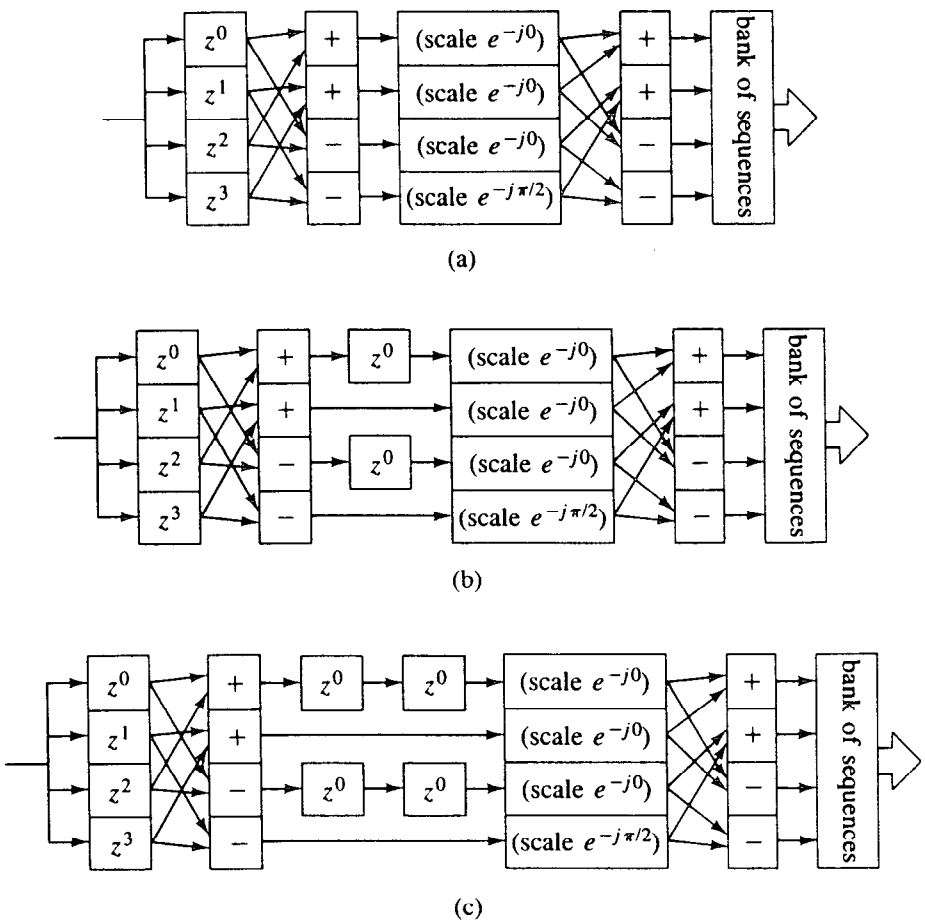


Figure 2.12 An example of increasing complexity resulting from equivalent-form manipulations.

As can be seen from this example, care must be taken to limit the complexity of the algorithms before they are used as starting points for further transformations. In ADE, simplifications are used to control the complexity of the signal processing expression. SIMPLIFICATION, when applied to a signal processing expression, returns the simplest direct description of the expression that the environment can find. The simplest description of a signal processing expression is obtained both by simplifying its subexpressions and by repeatedly simplifying the modified description. The actual simplifying transformations are encoded in ADE using over 300 of the 850 rules currently included.

The simplification process in ADE has two potential shortcomings. First, a good simplification can be missed because the steps required to generate the simplified form include steps that would cause subexpressions to not be in their simplified form. This restriction is imposed in order to prevent an unlimited potential growth in the number of expressions that must be considered. Second, and perhaps more fundamentally, ADE defines simplifications by a set of rules. It does not have any hooks for the user to define in what ways an expression is simpler than another one.

Hence, it is possible for ADE to "simplify" an expression to a form that is not appropriate.

Although we have pointed out that unconstrained algorithm manipulation has limitations, we must also point out that in many cases it is still a valuable tool. One such example is provided by the noninteger sampling rate conversion problem introduced in section 2.2. By applying unconstrained manipulations and simplifications to the straightforward implementation of a noninteger sampling rate conversion [shown in Figure 2.1(a)], ADE can automatically derive the computationally efficient implementation shown in Figure 2.13.[9] While the structure shown in Figure 2.13 can also be derived using the constrained manipulations, described in the next section, there are cases when efficient implementations cannot be derived using constrained manipulation. One such example is the efficient implementation of maximally decimated, octave band filters (Figure 2.14).

2.6 CONSTRAINED DERIVATION AND RANKING OF EQUIVALENT ALGORITHMS

The search for equivalent implementations of a signal processing expression must consider the equivalent implementations of the subexpressions as well as the complete expression itself. Since each of the subexpressions is independently manipulated and their equivalent forms are independently recombined to form new equivalent expressions, the size of the search space under consideration grows exponentially with the number of subexpressions. To illustrate, consider the problem of implementing the full FSK-code detector for 16 channels. Five independent descriptions of a simple, finite-length convolution are included in ADE: direct-form convolution, overlap-save convolution, the Fourier-domain representation of convolution, the z-domain representation of convolution, and the representation of convolution as the sum of scaled, shifted versions of the input. Thus, using these alternate forms as inputs into the incoherent summation, there will be $5^{16} \approx 10^{11}$ equivalent forms to consider. None of these implementations exploit the special structure of the modulated filter bank: the actual number of equivalent implementations that have to be considered is more than 10^{19}. Each of these implementations would then be reconsidered to see if any additional equivalent forms could be found, due to interactions between the implementations of the matched filters and the implementations of the incoherent processing. As illustrated by the projected size of the design space, some set of constraints must be imposed on the search process to avoid this exponential growth.

[9] Unconstrained manipulation was first shown to be effective by Myers [2] using a 2 : 3 noninteger sampling rate conversion in E-SPLICE. E-SPLICE generated a multirate structure of the same form as the one shown in Figure 2.13 for a 2 : 3 rate conversion. This demonstration of the potential of high-level signal processing compilation was made even more convincing by the subsequent publication of an independent article presenting this new type of polyphase structure [16]: E-SPLICE actually anticipated results from research in the area of noninteger sampling rate conversion algorithms.

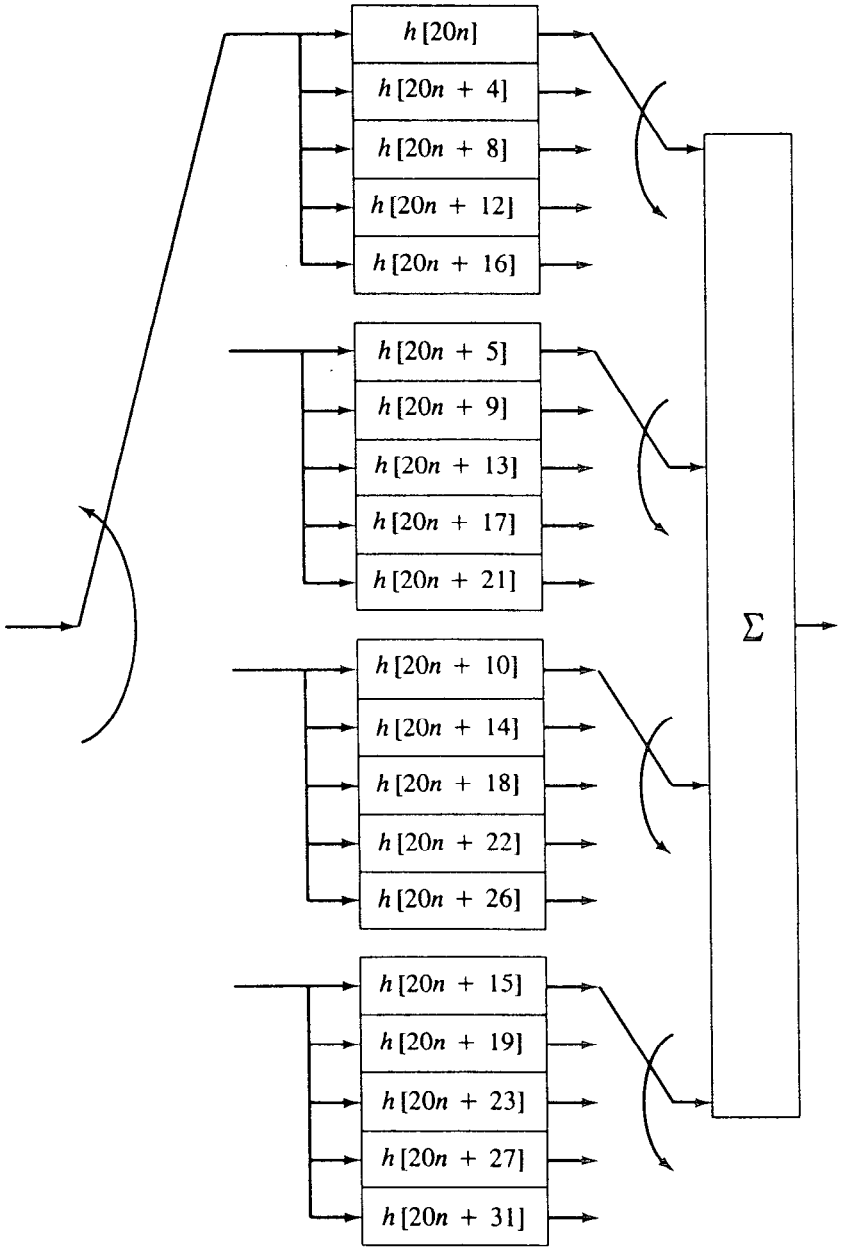
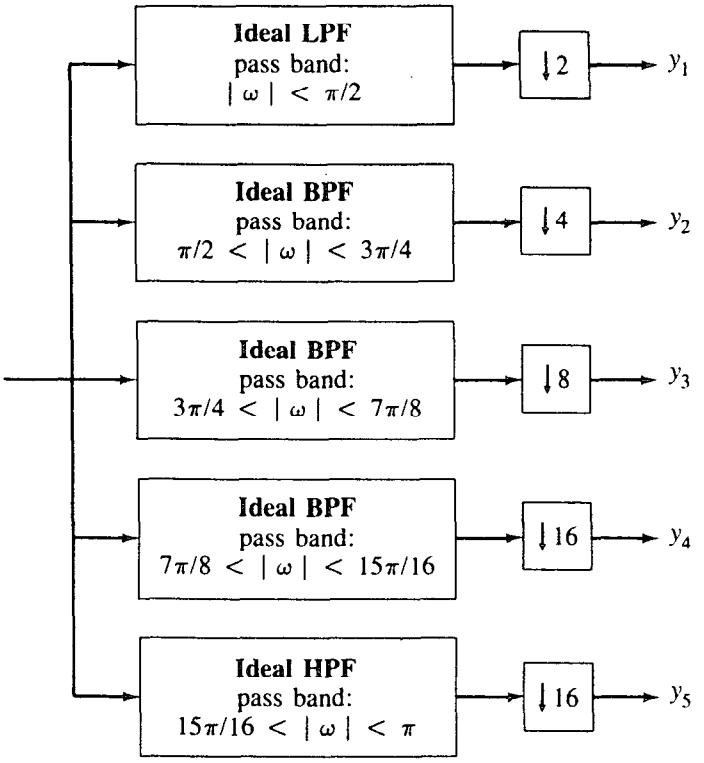


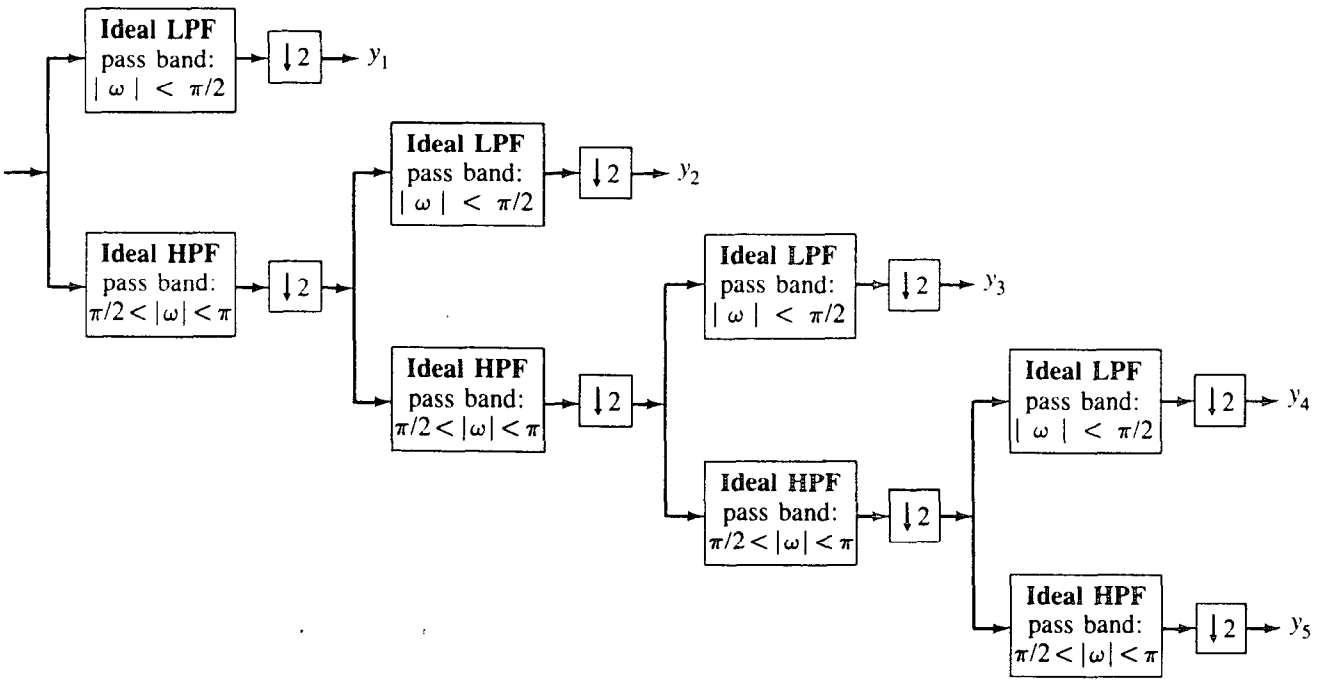
Figure 2.13 An efficient implementation of the 4:5 noninteger sampling rate conversion.

2.6.1 Approaches for Avoiding Exponential Growth of the Algorithm Design Space

One possible strategy for limiting the exponential growth in searches for efficient implementations relies on the cost measure of each subexpression to heuristically prune the space. Instead of enumerating all the equivalent implementations of a signal processing expression and then filtering out the inefficient and uncomputable structures using the overall cost measure, this strategy would immediately prune the



(a) Direct implementation of maximally decimated octave-band filters



(b) A more efficient implementation of maximally decimated octave band filters

number of subexpression implementations, prior to their upward propagation, based on their relative costs. This approach relies on the assumption that, when propagating two alternate implementations upward, the more expensive implementation will not be incorporated into any of the efficient implementations of the enclosing expression. Unfortunately, this pruning strategy suffers from the interaction of subexpression costs: the cost of using one implementation of a subexpression is often ameliorated by reusing part or all of the subexpression in some other part of the enclosing expression. For example, the computational savings of the FFT-based STFT results from the interaction of the computational cost of the subexpressions: it is the reuse of the partial summations which reduces its computational cost to $O(N \log N)$. Thus, the contribution of a subexpression to the overall cost of an enclosing expression is not independent of the other parts of the enclosing expression.

The approach that ADE takes to limiting the search space is to attempt to exploit the internal regularity of signal processing algorithms. Signal processing algorithms are often described at different levels of detail. For example, the four-point, rectangularly windowed, short-time Fourier transform of a sequence can be described by either of the structures shown in Figure 2.15. The structure in Figure 2.15(a) is by definition the STFT. Figure 2.15(b) shows a fully expanded short-time FFT structure, i.e., one in which common subexpressions are not combined. Starting from the high-level description of an algorithm, the regularity in the low-level computational structure can often be asserted. For example, when the SHORT-TIME-FT system is expanded into the structure in Figure 2.15(b), the underlying regularity inherent in Figure 2.15(b) can be noted. By enforcing these internal correspondences in the low-level descriptions, the space of equivalent forms that is explored can be drastically reduced. This approach to pruning the search is heuristic. However, the regularity of the computation suggests that the efficient implementations will reflect the same regularity: if separate sections of an algorithm are very similar, then the efficient implementations of these separate sections are likely to coincide.

To illustrate what is meant by internal regularity within an algorithm, consider the description of the short-time Fourier transform given in Figure 2.15(b). This

Figure 2.14 Implementation of maximally decimated, ideal, octave-band filters. Maximally decimated octave-band filters are useful in speech and image coding as well as perceptual modeling. A simple and direct implementation of maximally decimated, octave-band filters, using ideal low- and band-pass filters, is shown in part (a) of this figure. An implementation that is more computationally efficient is shown in part (b): here the band-pass filters in the octaves above base-band have been divided into cascades of high-pass filters, followed by decimation, followed by another stage of high- and low-pass filters. This implementation is one example of a computationally efficient implementation which would not be found using constrained manipulation as it is described in section 2.6 of this chapter. The reason that this implementation would be missed using constrained manipulation is that each branch of the algorithm is treated differently: the branch that computes y_1 is unchanged, the branch that computes y_2 is separated into two stages of filtering/decimation, the branch that computes y_3 is separated into three stages of filtering/decimation, and so on.

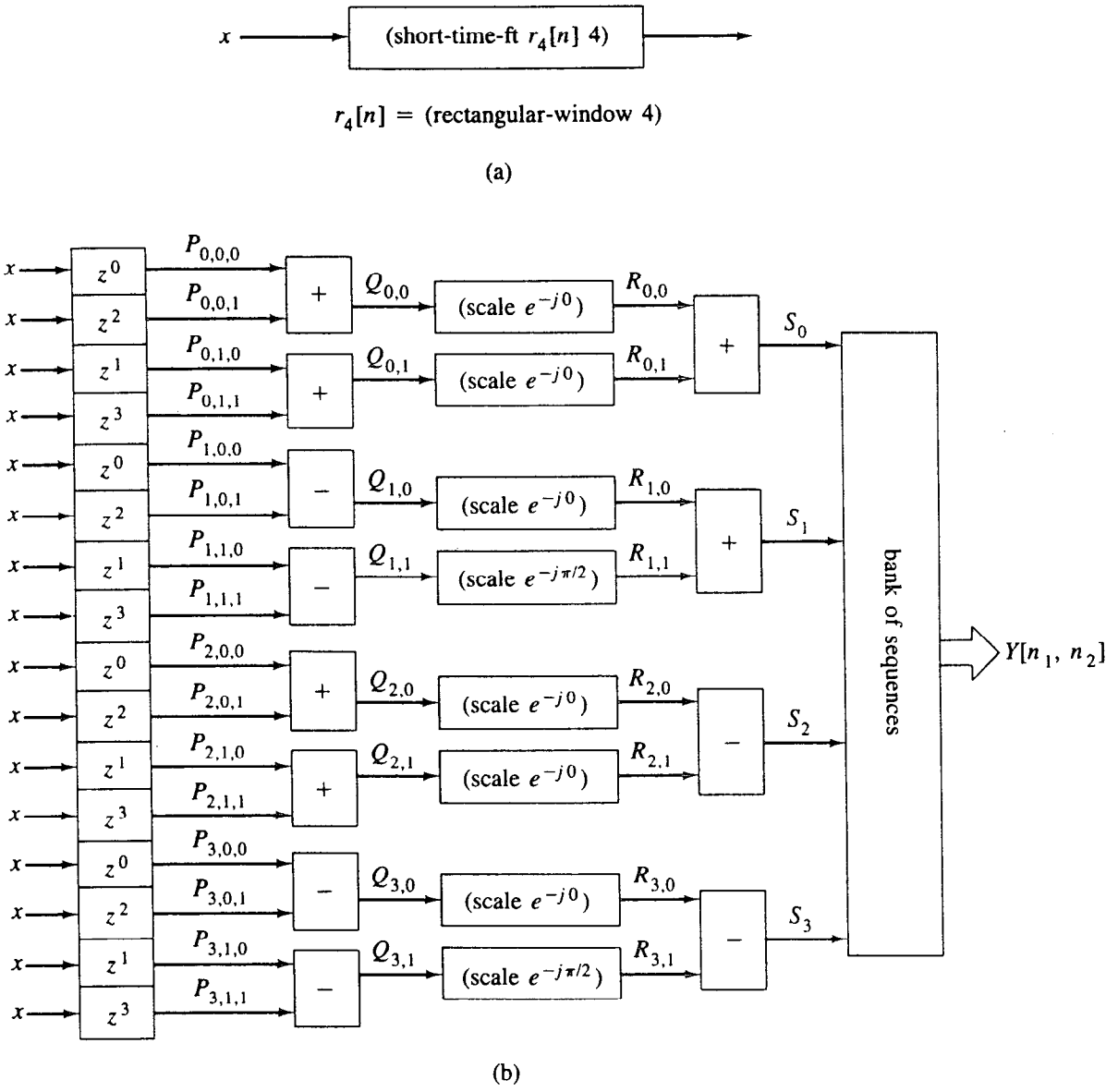


Figure 2.15 Two alternate descriptions of a four-point, rectangularly windowed short-time Fourier transform.

implementation of the short-time Fourier transform is provided explicitly by one of the transformation rules included in the definition of SHORT-TIME-FT. The regularity of this implementation is also explicitly noted by the rule. In particular, the similarity of the sequences feeding into the BANK-OF-SEQUENCES is pointed out using a "correspondence constraint": by placing a correspondence constraint on the subexpressions feeding into the BANK-OF-SEQUENCES, the similarity of these subexpressions is recorded. As will be discussed later in this section, a correspondence constraint forces the similar subexpressions to be subject to a single series of transformation rules, so that, if a transformation is applied to one of a set of similar expressions, that transformation must also be applied to all the other members of the set.

In addition to the point of similarity at the BANK-OF-SEQUENCES, the structure shown in Figure 2.15(b) has two other levels of similarity at the inputs to the addition/subtraction systems: at each of the two butterfly stages, the first addend into the kth butterfly is similar to the second addend into the same kth butterfly. Thus, more correspondence constraints are placed on the inputs into each of these systems. Through these correspondence constraints, the similarity of the corresponding subexpressions is explicitly noted, resulting conceptually in the manipulation of $Y[n_1, n_2]$, $S_k[n]$, $R_{k,l}[n]$, $Q_{k,l}[n]$, and $P_{k,l,m}[n]$ where

$$\begin{aligned} Y[n_1, n_2] &= S_{n_2[n_1]} \\ S_k[n] &= R_{k,0}[n] \pm R_{k,1}[n] \\ R_{k,l}[n] &= e^{-j(2\pi/4)lk} Q_{k,l}[n] \\ Q_{k,l}[n] &= P_{k,l,0}[n] \pm P_{k,l,1}[n] \\ P_{k,l,m}[n] &= x[n + l + 2m] \end{aligned}$$

By enforcing these constraints, the rearranged algorithms will also have a regular internal structure and the number of independently manipulated subexpressions is reduced, in this case from $O(N^2)$ to $O(\log N)$.

2.6.2 Algorithmic Transformations in the Presence of Correspondence Constraints

This subsection examines the process of finding equivalent forms of an algorithm in the presence of correspondence constraints.

When a correspondence constraint is imposed, it is imposed on the inputs to a system. For example, a correspondence constraint was imposed on the inputs to the BANK-OF-SEQUENCES in Figure 2.15(b), creating nine sets of similar sequences: S_k, $R_{k,0}$, $R_{k,1}$, $Q_{k,0}$, $Q_{k,1}$, $P_{k,0,0}$, $P_{k,0,1}$, $P_{k,1,0}$, and $P_{k,1,1}$. Additional correspondence constraints were also imposed on the inputs for S_k, reducing the number of separately manipulated sets of signals to five (S_k, $R_{k,l}$, $Q_{k,l}$, $P_{k,l,0}$, and $P_{k,l,1}$) and on the inputs for $Q_{k,l}$, further reducing the number of separately manipulated sets to four (S_k, $R_{k,l}$, $Q_{k,l}$, and $P_{k,l,m}$). As mentioned above, imposing these constraints forces the similar signals to be manipulated as a set: the transformation rules are applied to the set of similar signals, instead of just to the individual signals. To continue the STFT example, if the rule shown on lines 13–30 of Figure 2.10 is applied to one of the sequences labeled $Q_{k,l}$, then this same rule must be applied to all the other sequences labeled $Q_{k,l}$. The remainder of this subsection considers the changes required in the recursive search and subexpression manipulation, described in section 2.5, to accommodate these requirements.

The approach to finding equivalent forms described in section 2.5 involved both recursive search and subexpression manipulation. The recursive search, in which newly generated equivalent forms act as the starting point for further transformations, is used without modification in searches for constrained equivalent

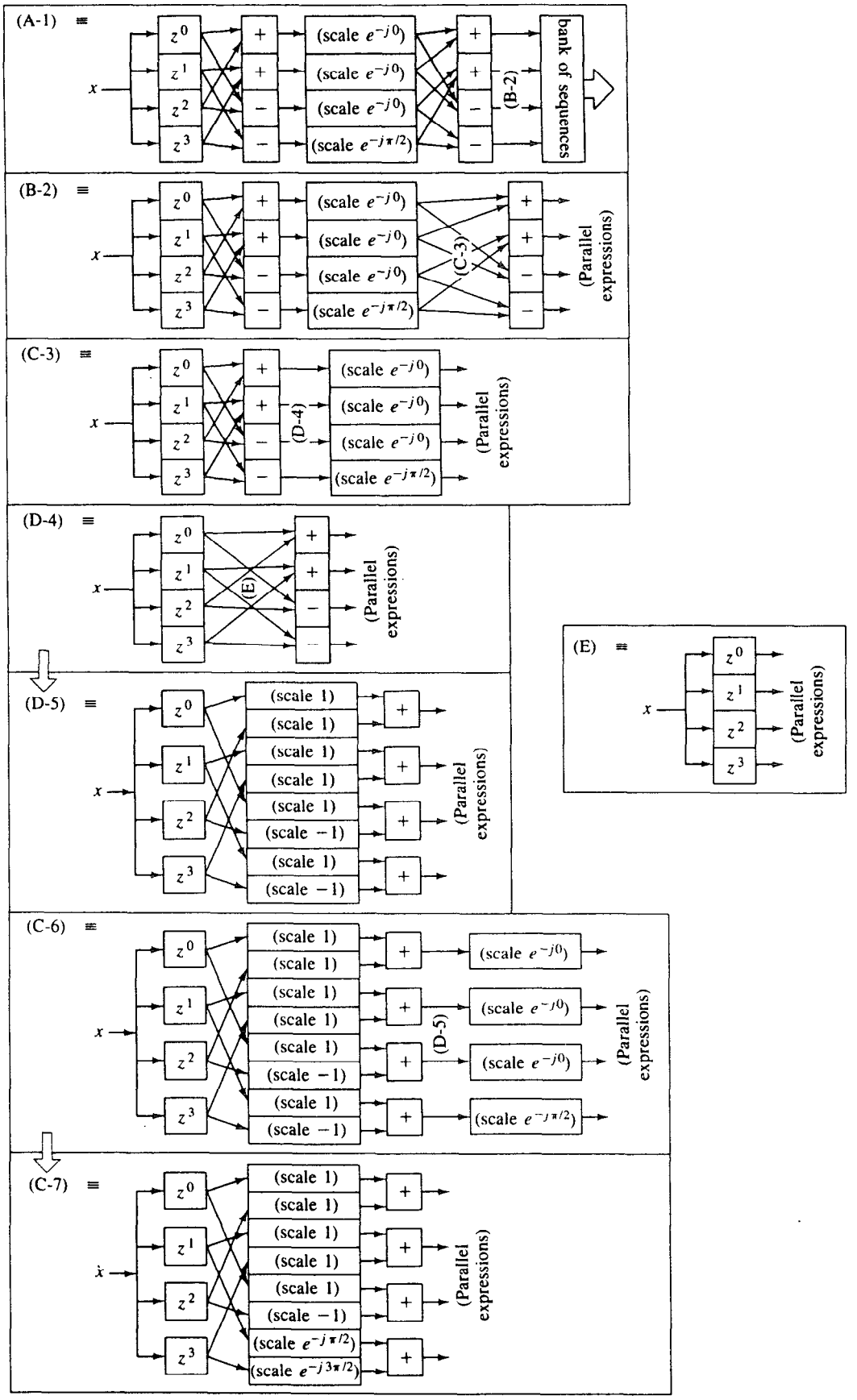


Figure 2.16 (continued)

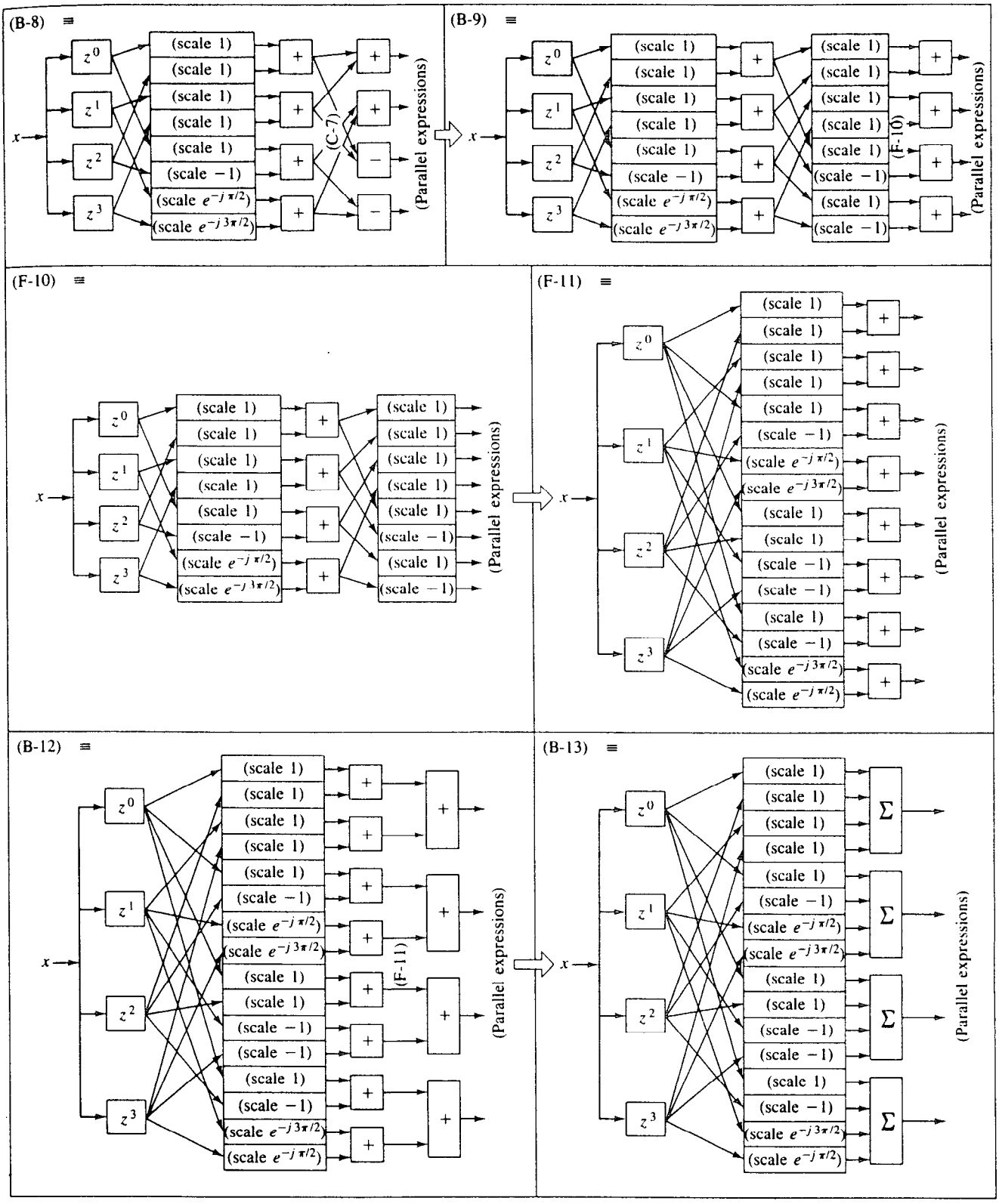


Figure 2.16 (continued)

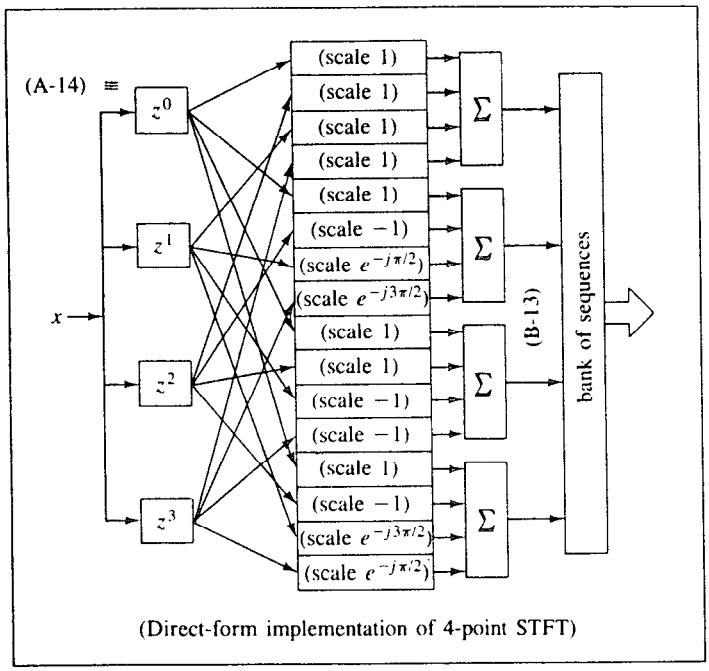


Figure 2.16 (continued)

D-4. The structure D-4 is transformed into the structure D-5 and this is used to replace subexpression D-4 in the expression C-3, resulting in the structure C-6. The structure C-6 is then transformed into the structure C-7, which then replaces subexpression C-3 in the expression B-2, resulting in the structure B-8. The structure B-8 is transformed into the structure B-9. The subexpression F-10 is then extracted from the expression B-9, manipulated into subexpression F-11, and replaced back into the expression B-9, resulting in the structure B-12. Next, the structure B-12 is transformed into the structure B-13 and this structure replaces subexpression B-2 in the expression A-1, yielding the structure A-14 as the final result.

As mentioned earlier, the actual transformation rules that are used to find constrained equivalent forms and constrained simplifications are the same as those used to find unconstrained equivalent forms and unconstrained simplifications, respectively. It is the manner in which these transformations are combined that provides the distinction between the constrained and unconstrained searches. In particular, in unconstrained searches, all subexpressions are manipulated independently while in constrained searches, similar subexpressions are manipulated as sets.

As a measure of the importance of imposing correspondence constraints, we note that for the modulated filter bank example of section 2.4, the number of possible structures using unconstrained manipulation is more than 10^{19} for the rectangular-window matched filters and it is more than 10^{58} for the Hanning-window matched filters. Using constrained manipulations, the numbers are 13 and 20, respectively.[10] Thus, the use of regularity constraints is essential for solving these problems.

[10] The large number of structures generated by unconstrained manipulation is a result of considering all possible combinations of the alternate implementations of the subexpressions. Since there are 16 branches with 16 inputs each, the number of possible combinations grows very quickly.

2.6.3 Propagating Correspondence Constraints to a Modified Structure

When constrained expressions are manipulated, new expressions are often generated on which the same correspondence constraints should be imposed. To illustrate, consider the manipulations shown in Figure 2.16. The result of these manipulations is a direct-form implementation of the short-time Fourier transform (node A-14). In order to reflect the correspondence constraints of the original structure, this new bank of sequences should also include two correspondence constraints: the inputs to the bank of sequences should be constrained to coincide as should the inputs to the addition systems. Unless these constraints are imposed, this new form will introduce unconstrained structures into the constrained manipulations. ADE propagates structural constraints to new expressions automatically. When a constrained expression is manipulated, the inputs that are constrained to be parallel are noted prior to manipulation. After each manipulation of a constrained structure, ADE attempts to impose the analogous correspondence constraints on the modified structure.

2.7 CONTRIBUTIONS AND LIMITATIONS

Our goals in this chapter were to describe the mechanisms by which an integrated signal processing environment could manipulate signal processing algorithms and to demonstrate the potential of these manipulations. ADE has been used within this chapter to demonstrate this potential.

ADE makes extensive use of general signals, rules, and regularity constraints. General or abstract signals provide the variables in the manipulation of algorithms while the rules in ADE about signal processing provide the information required to manipulate this "signal processing algebra." Regularity constraints limit the exponential expansion that manipulation of subexpressions introduces by exploiting the internal regularity of signal processing algorithms to limit the size of the explored search space. This regularity in the low-level signal processing descriptions is noted using information provided by the higher level description of the same operation. Without these constraints, many FFT-based and polyphase-based algorithms would be beyond the scope of consideration, due to exponential expansion of these design spaces.

ADE demonstrates the potential of integrated signal processing environments. Its algorithm rearrangement capabilities have been used to generate innovative, computationally efficient implementations in two well-developed areas of signal processing. However, much work remains to be done to transform the concepts embodied in ADE into practical signal processing workstations:

- The user interface should be made more accessible and pleasant.
- A fast algorithm-rearrangement facility, providing only well-known algorithm implementations, should be included as an alternative to the current rearrange-

ment facilities which do constrained searches of the full design space. This facility would provide quick compilations by ignoring the possibility of efficient nonconventional implementations. An example of this facility would be one that would provide the classic implementations of a noninteger sampling rate conversion [e.g., Figures 2.1(b) and (c)] without exploring the full design space. This facility would miss the more nonconventional implementations (e.g., Figure 2.13).

- The derivation of explicitly recursive algorithms, such as the one shown in (2.3)–(2.6), is an unexplored area of research.
- Regularity within expressions should be automatically detected. In ADE, the regularity of a signal processing algorithm must be explicitly pointed out. Although propagation of the regularity constraints, both within the algorithm and to modified expressions, is supported by the environment, the initial description of the constraints must be done manually.

As can be seen by the breadth of this partial list of "things to do," much work still remains to be done to achieve a signal processing environment that facilitates the algorithm manipulation as well as numeric processing of signals and algorithm definition. However, as asserted previously, the rewards for research in this field have thus far been high.

ACKNOWLEDGMENTS

We would like to acknowledge the contributions to this work by Professor Hal Abelson, Professor Randy Davis, Dr. Webster Dove, Dr. Robert Kahn, Dr. Evangelos Milios, Dr. Douglas Mook, Dr. Bruce Musicus and Professor Victor Zue. We would also like to thank Dr. Gary Kopec, Dr. Evangelos Milios and Dr. Malcolm Slaney for their comments and help with this chapter. This work was supported in part by the Advanced Research Projects Agency, Lockheed Sanders, Inc., the AMOCO Foundation, and Schlumberger-Doll Research.

REFERENCES

[1] M. M. Covell, "An Algorithm Design Environment for Signal Processing," RLE Technical Report 549, Cambridge, Mass.: MIT (1989).

[2] C. S. Myers, "Signal Representation for Symbolic and Numerical Processing," RLE Technical Report 521, Cambridge, Mass.: MIT (1986).

[3] G. E. Kopec, "The Signal Representation Language SRL," *IEEE Trans. Acoustics, Speech, and Signal Processing*, 33 (August 1985), 921–32.

[4] W. P. Dove, C. S. Myers, and E. E. Milios, "An Object-Oriented Signal Processing Environment: The Knowledge-Based Signal Processing Package," RLE Technical Report 502, Cambridge, Mass.: MIT (1984).

[5] R. E. Crochiere and L. R. Rabiner, *Multirate Digital Signal Processing* (Englewood Cliffs, N.J.: Prentice Hall, 1983).

[6] J. S. Jaffe and J. M. Richardson, "A Code-Division Multiple Beam Imaging System," in *Proc. OCEANS '89* (1989):1015–20.

[7] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing* (Englewood Cliffs, N.J.: Prentice Hall, 1989).

[8] P. M. Peterson and J. A. Frisbie, "Interactive Environment for Signal Processing on a VAX Computer," in *Proc. ICASSP'87* (1987):1891–93.

[9] W. P. Dove, "Knowledge-based Pitch Detection," RLE Technical Report 518, Cambridge, Mass.: MIT (1986).

[10] E. Sacks, "Qualitative Mathematical Reasoning," in *Proc. of the Int. Joint Conference on Artificial Intelligence* (1985):137–39.

[11] The Mathlab Group, *MACSYMA Reference Manual*, Lab. for Computer Science, Cambridge, Mass.: MIT (1983).

[12] Symbolics, Inc., *Symbolics Common Lisp: Language Concepts*, Cambridge, Mass.: Symbolics, Inc. (1986).

[13] R. C. Singleton, "An Algorithm for Computing the Mixed Radix Fast Fourier Transform," *IEE Trans. Audio and Electroacoustics* 17 (June 1969):93–103.

[14] J. D. Markel, "FFT Pruning," *IEEE Trans Audio and Electroacoustics* 19 (December 1971):305–11.

[15] D. P. Skinner, "Pruning the Decimation in Time FFT Algorithm," *IEEE Trans. Acoustics, Speech and Signal Processing*, 34 (April 1976):305–11.

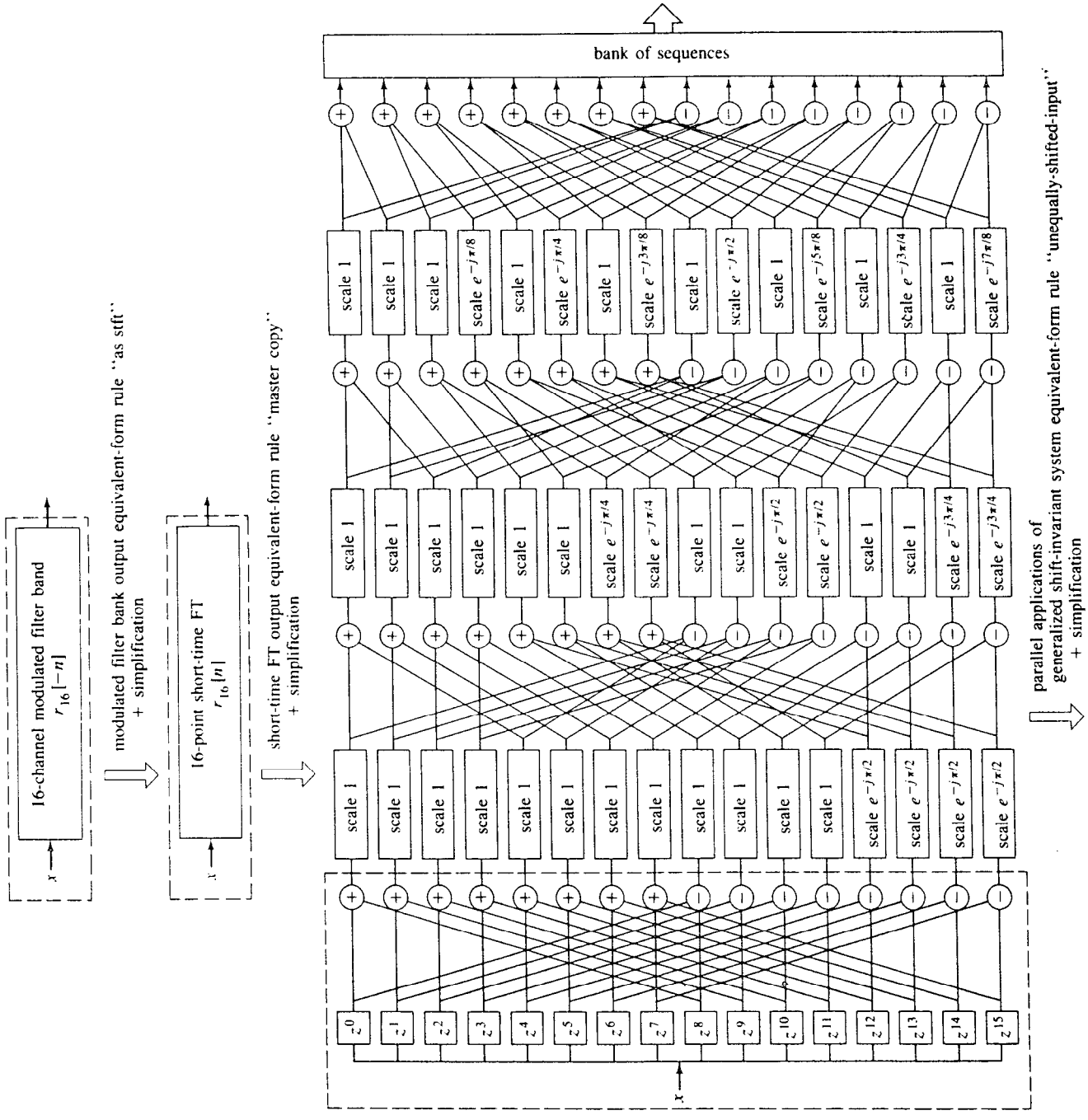
[16] C.-C. Hsiao, "Polyphase Filter Matrix for Rational Sampling Rate Conversions," in *Proc. ICASSP'87* (1987):1056–59.

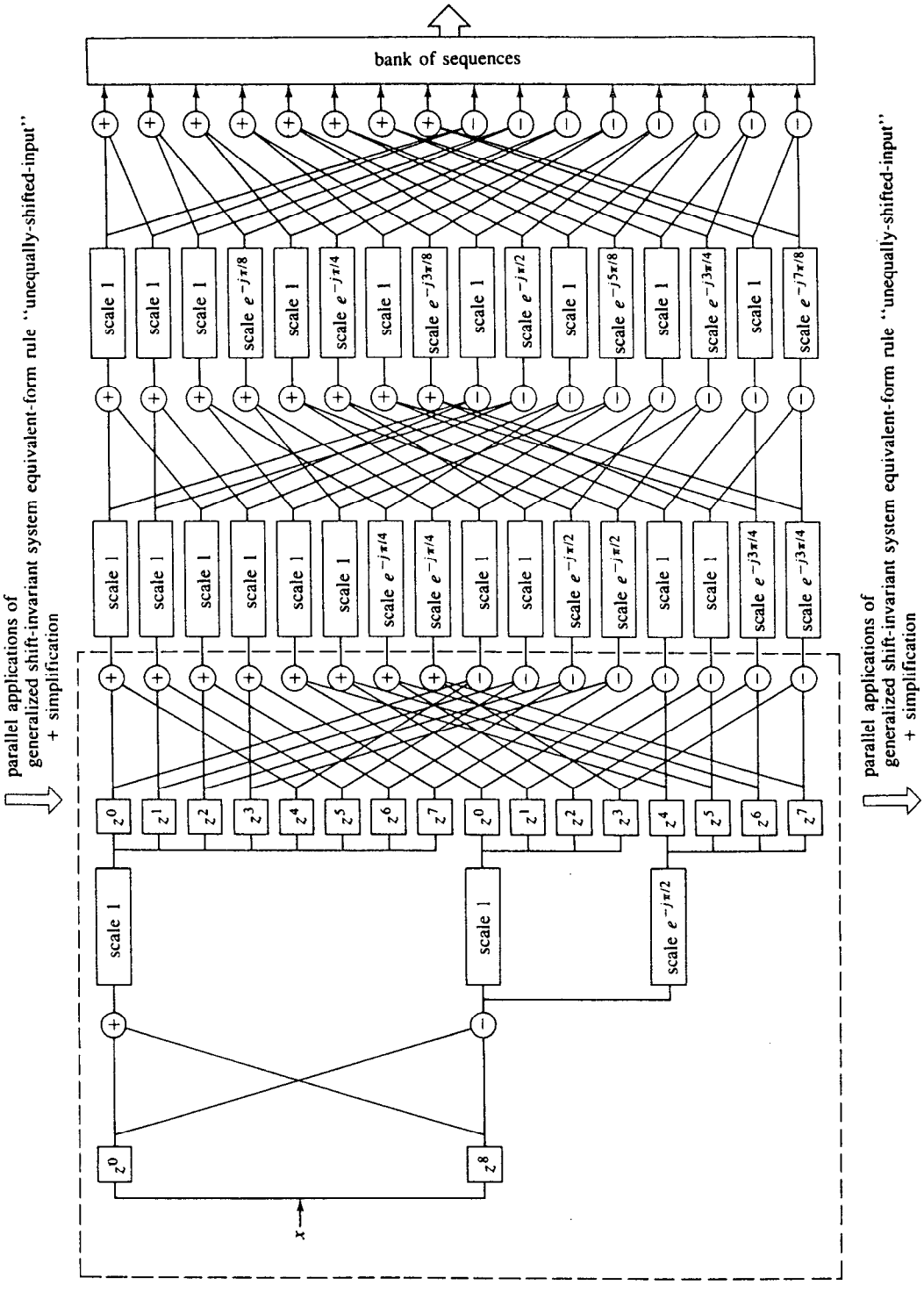
[17] MathWorks, Inc., *MATLAB: User's Guide*, Natick, Mass.: The MathWorks, Inc. (1989).

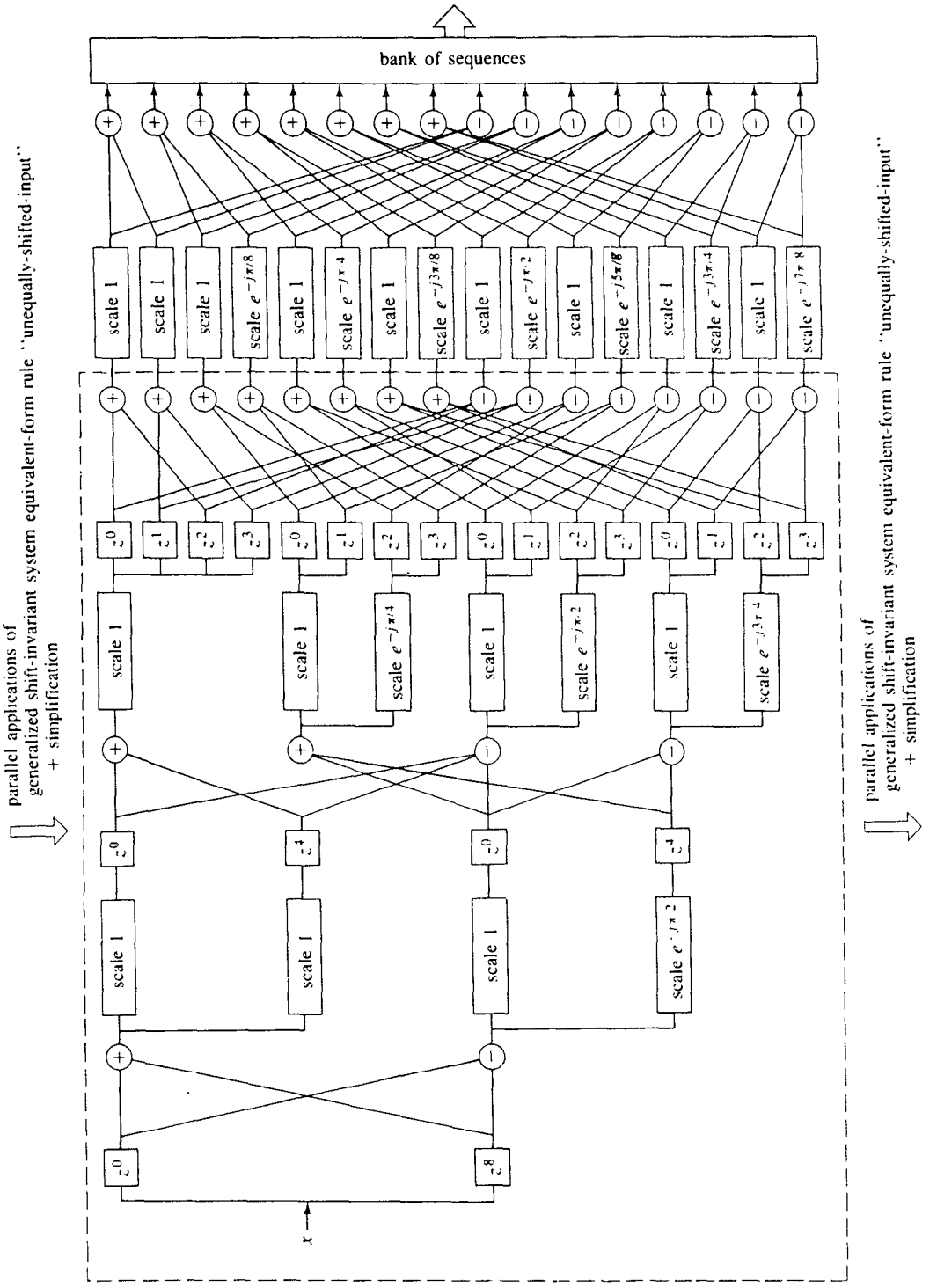
[18] Signal Technology, Inc., *ILS: Interactive Laboratory System*, Goleta, Calif.

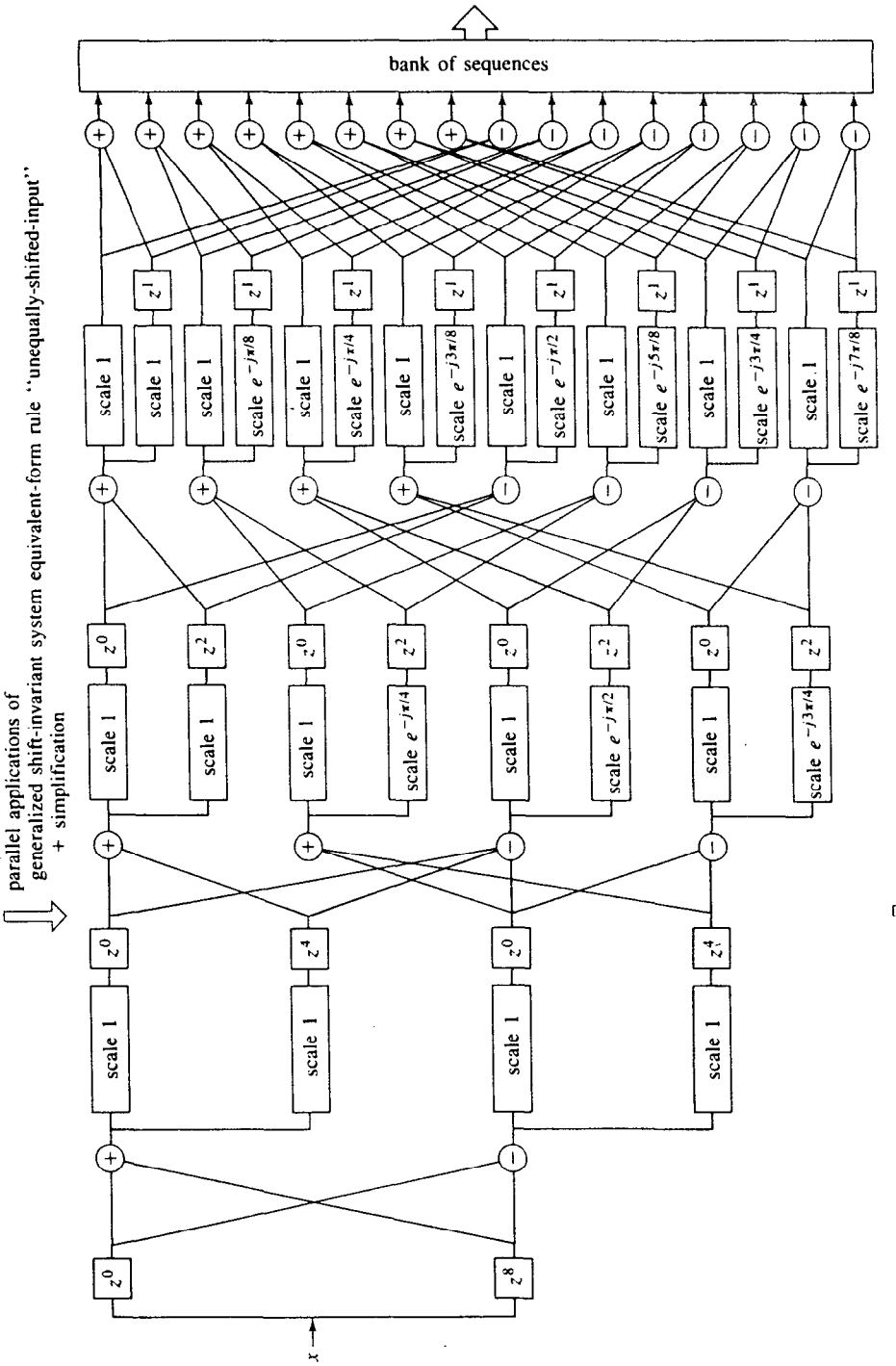
APPENDICES

Appendix A.1 The sequence of transformations used in going from the 16-channel rectangular-window modulated filter bank to the pruned FFT structure shown in Figure 2.7.



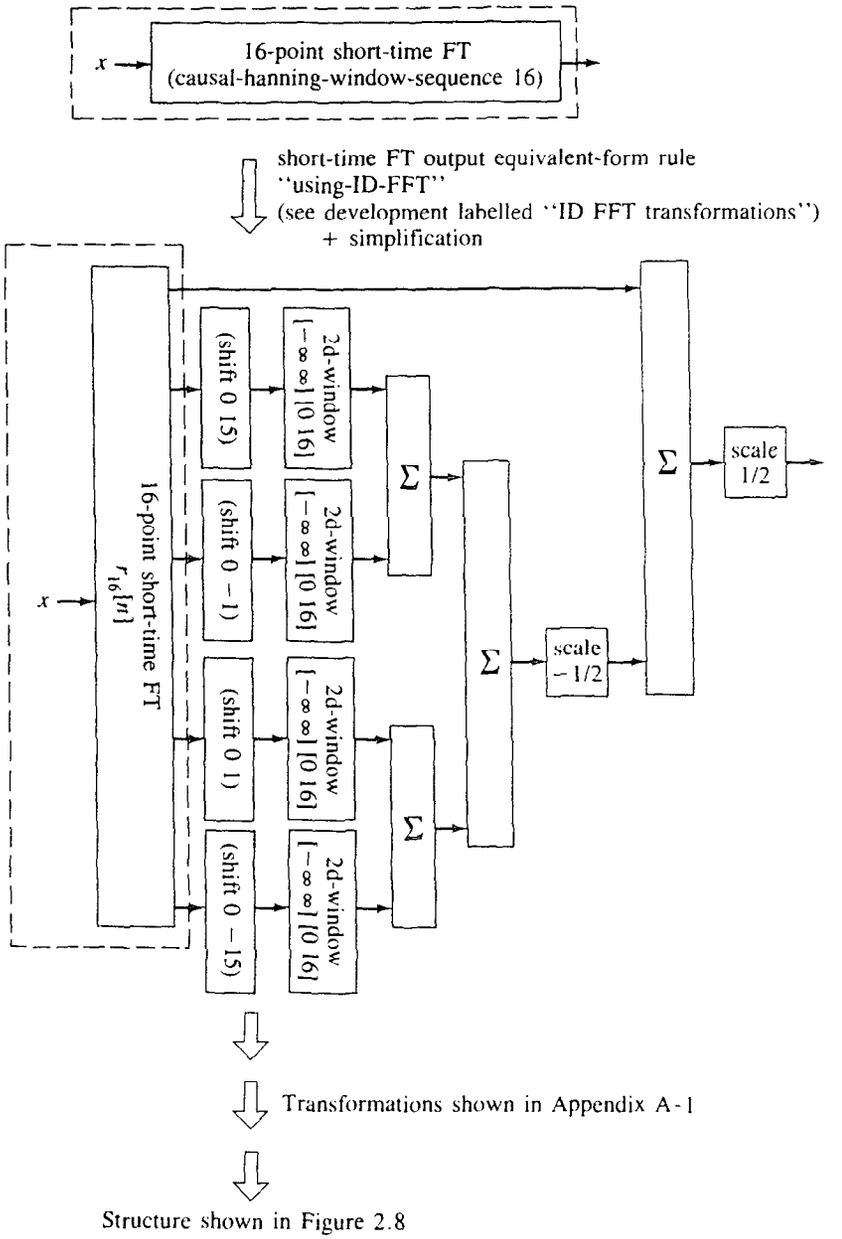

bank of sequences
parallel applications of
generalized shift-invariant system equivalent-form rule "unequally-shifted-input"
+ simplification
parallel applications of
generalized shift-invariant system equivalent-form rule "unequally-shifted-input"
+ simplification
scale 1
scale $e^{-j\pi/4}$
scale $e^{-j3\pi/4}$
scale $e^{-j\pi/2}$
scale $e^{-j5\pi/8}$
scale $e^{-j3\pi/8}$
scale $e^{-j7\pi/8}$
z^0
z^1
z^2
z^3
z^4
z^5
z^6
z^7
z^8
x


bank of sequences
parallel applications of generalized shift-invariant system equivalent-form rule "unequally-shifted-input" + simplification
parallel applications of generalized shift-invariant system equivalent-form rule "unequally-shifted-input" + simplification
scale 1
x



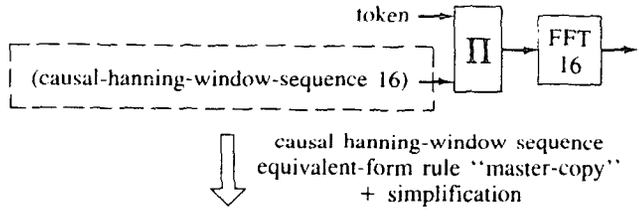
Pruned FFT structure shown in Figure 2.7

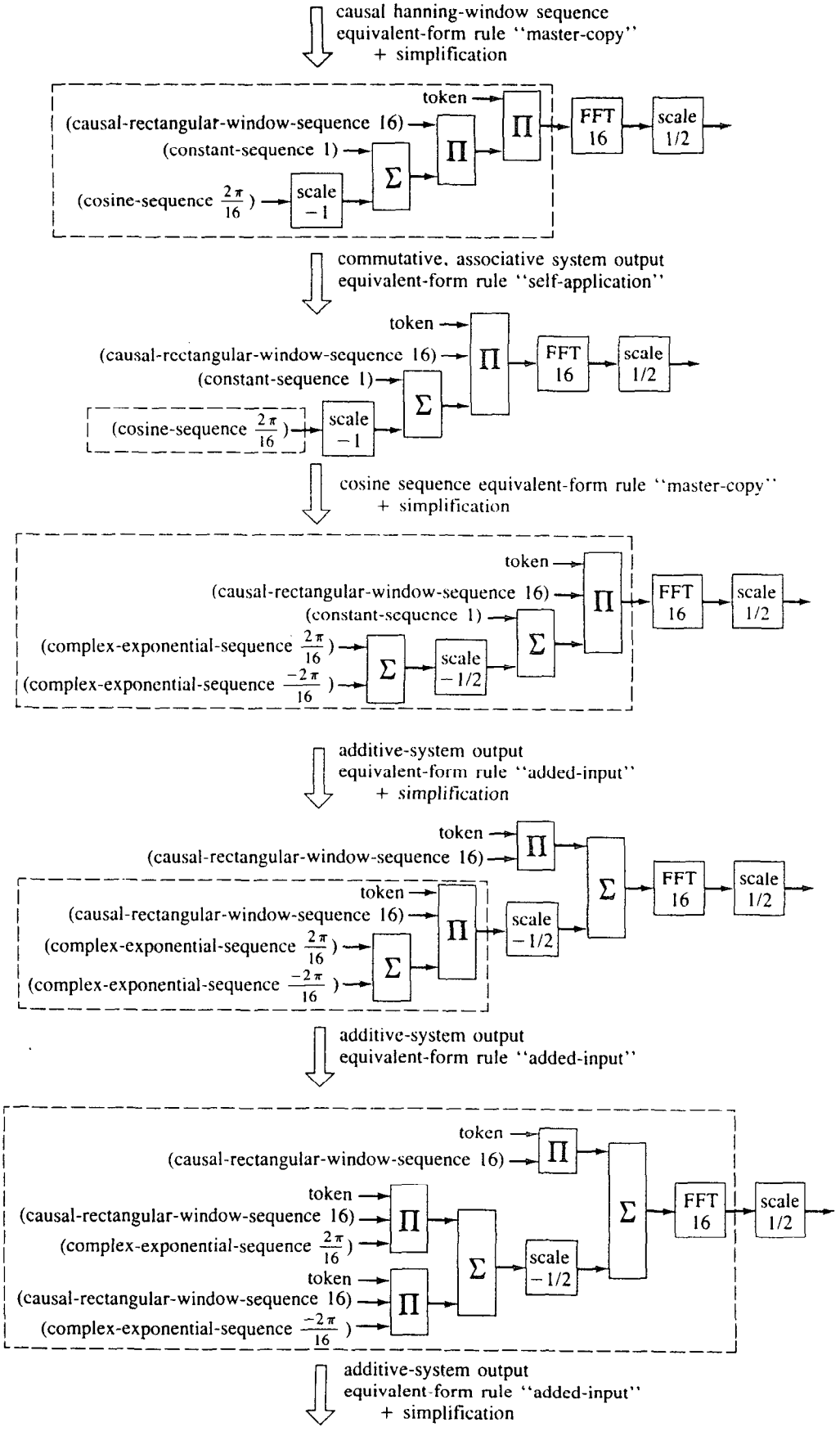
Appendix A.2 The sequence of transformations used in going from the 16-point short-time Fourier transform with a 16-point Hanning window to the structure shown in Figure 2.8.

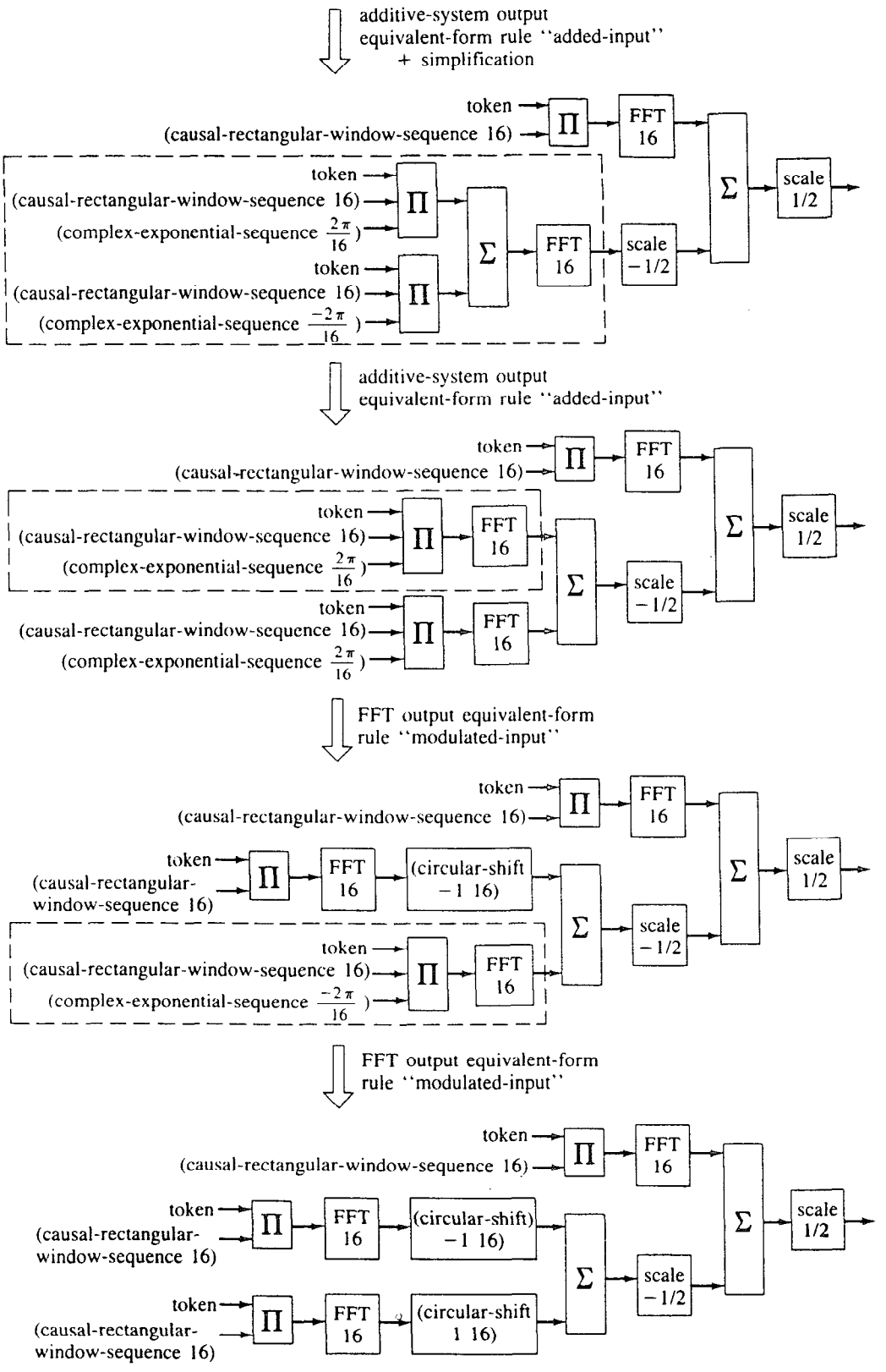


ID FFT transformations

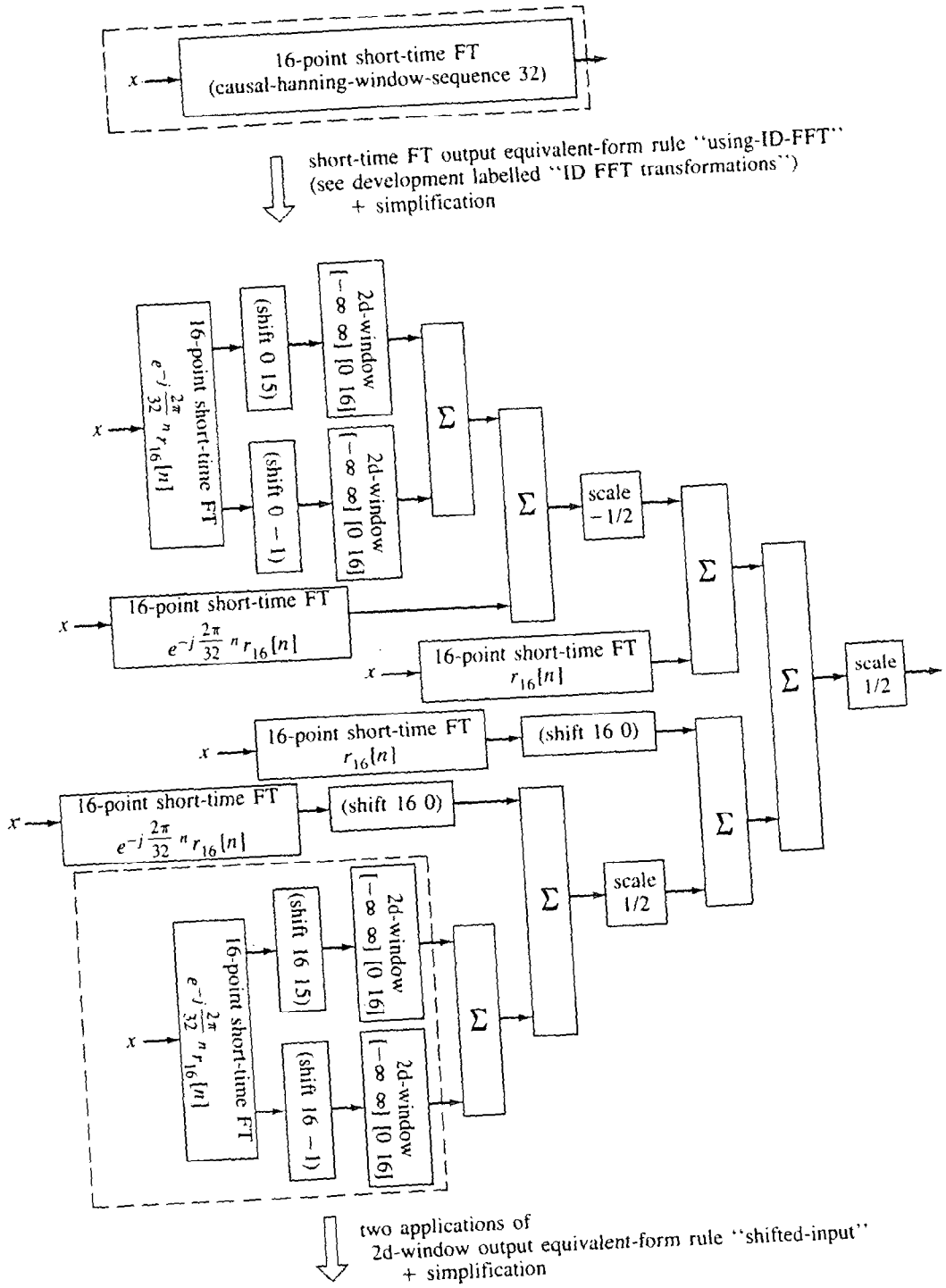
Let "token" represent the abstract discrete-time sequence generated by the short-time Fourier transform output equivalent-form rule "using-1d-fft"

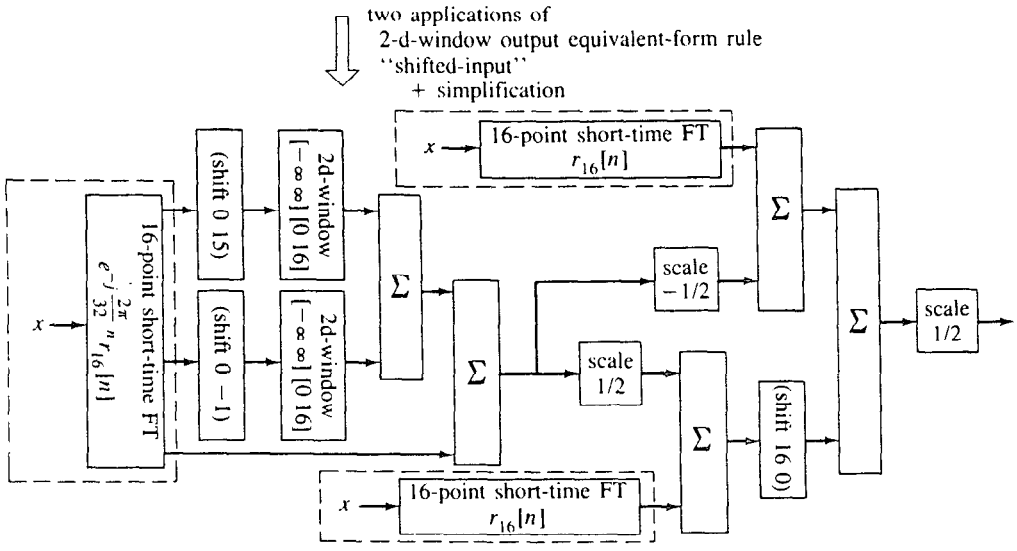



causal hanning-window sequence
equivalent-form rule "master-copy"
+ simplification
token
(causal-rectangular-window-sequence 16)
(constant-sequence 1)
(cosine-sequence $\frac{2\pi}{16}$)
scale −1
Σ
Π
FFT 16
scale 1/2
commutative, associative system output
equivalent-form rule "self-application"
cosine sequence equivalent-form rule "master-copy"
+ simplification
(complex-exponential-sequence $\frac{2\pi}{16}$)
(complex-exponential-sequence $\frac{-2\pi}{16}$)
scale −1/2
additive-system output
equivalent-form rule "added-input"
+ simplification


additive-system output
equivalent-form rule "added-input"
+ simplification
token
(causal-rectangular-window-sequence 16)
Π
FFT 16
Σ
scale 1/2
(complex-exponential-sequence 2π/16)
(complex-exponential-sequence −2π/16)
scale −1/2
additive-system output
equivalent-form rule "added-input"
FFT output equivalent-form
rule "modulated-input"
(causal-rectangular-
window-sequence 16)
(circular-shift −1 16)
FFT output equivalent-form
rule "modulated-input"
(circular-shift
1 16)

Appendix A.3 The sequence of transformations used in going from the 16-point short-time Fourier transform with a 32-point Hanning window to the structure shown in Figure 2.9.





multiple applications of
short-time FT output equivalent-form rule "master copy"
+ simplification

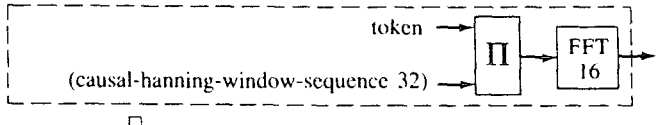
multiple applications of
generalized homogeneous system equivalent-form rule "unequally-scaled-input"
+ simplification

multiple applications of
generalized shift-invariant system equivalent-form rule "unequally-shifted-input"
+ simplification

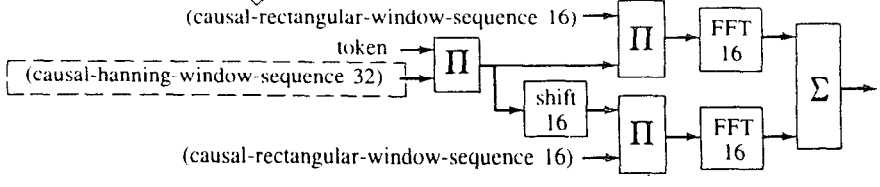
structure shown in Figure 2.9 (a)

ID FFT transformations

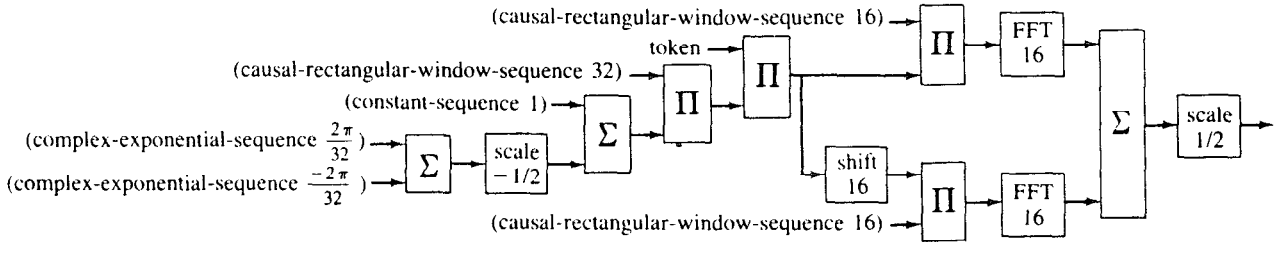
Let "token" represent the abstract discrete-time sequence generated by the short-time Fourier transform output equivalent-form rule "using-1d-fft"



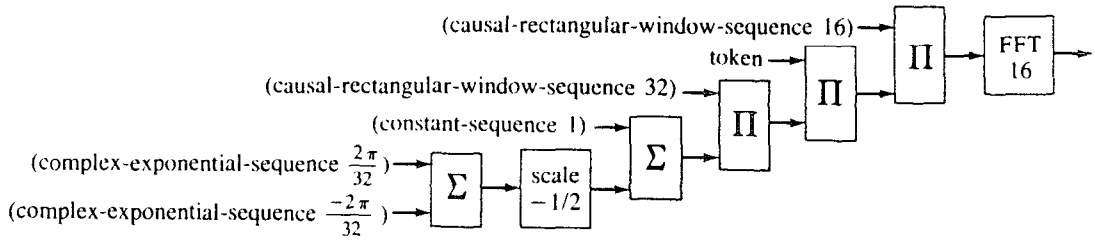
FFT output equivalent-form rule
"aliased-input"



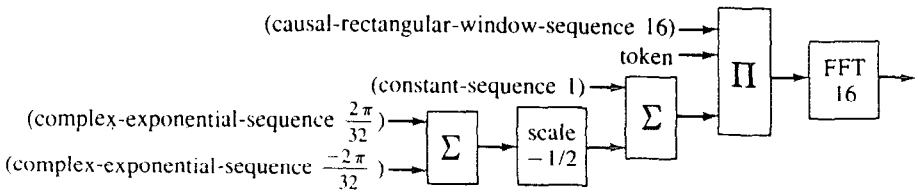
causal hanning-window sequence equivalent-form rule "master-copy"
+ simplification
cosine sequence equivalent-form rule "master copy"
+ simplification



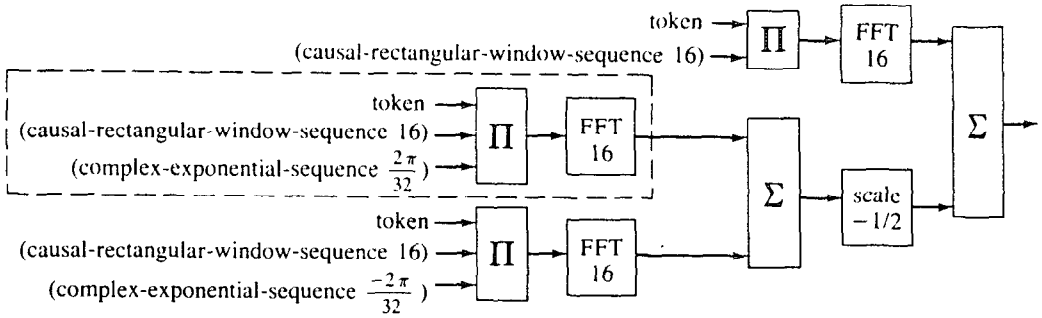
ID FFT transformations continued: manipulation of structure below



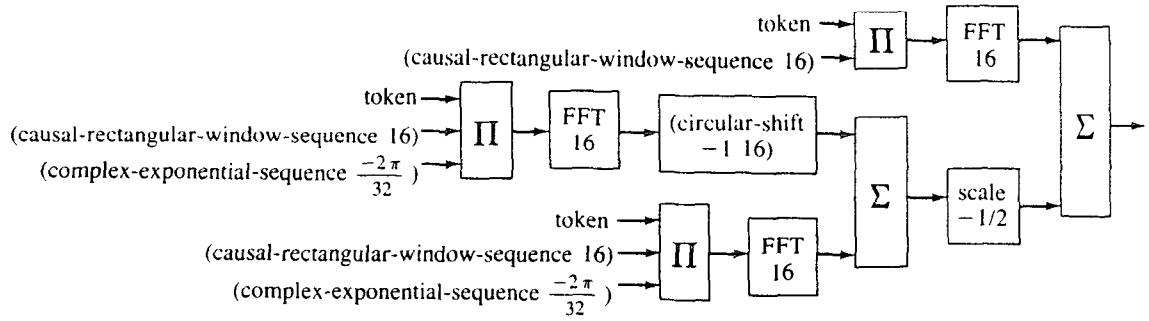
two applications of
commutative, associative system output equivalent-form rule
"self-application"
+ simplification



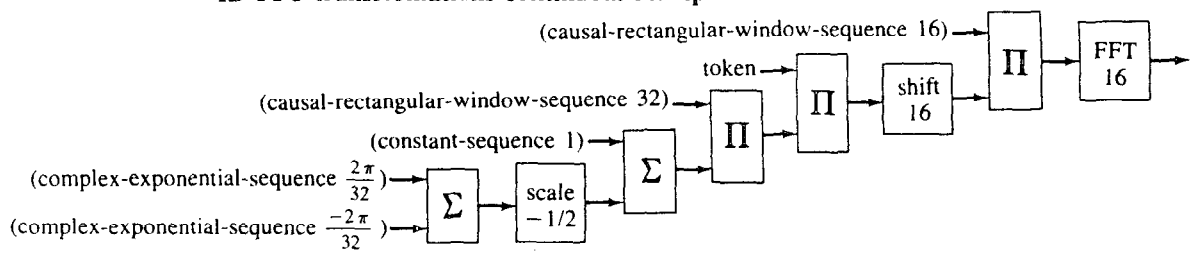
four applications of
additive-system output equivalent-form rule
"added-inputs"
with simplification between



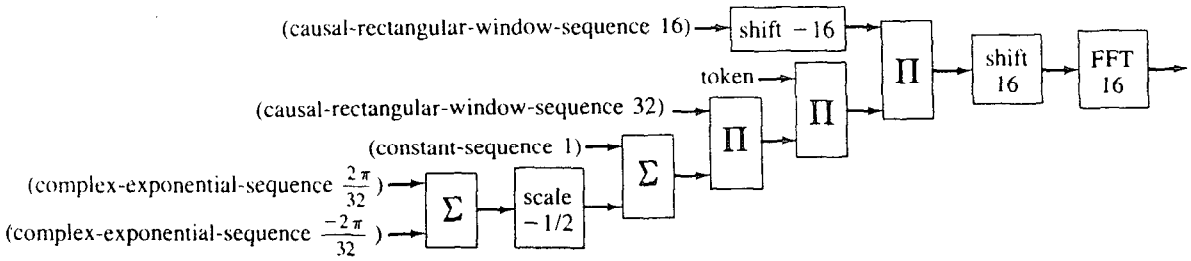
FFT output equivalent-form rule
"modulated-input"



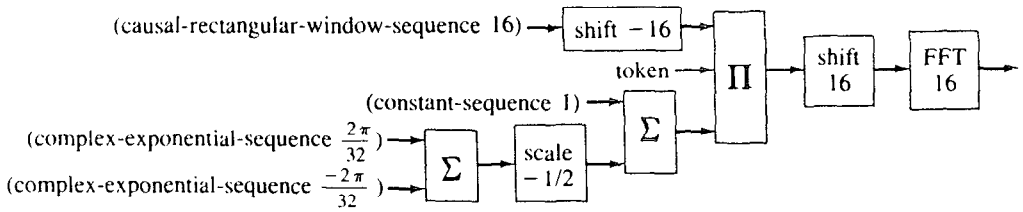
ID FFT transformations continued: manipulation of structure below



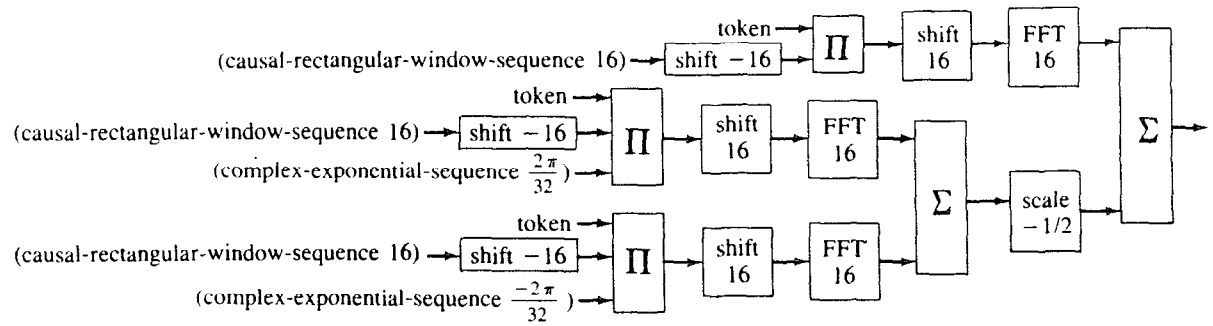
generalized shift-invariant system output equivalent-form rule "single-shifted-input"



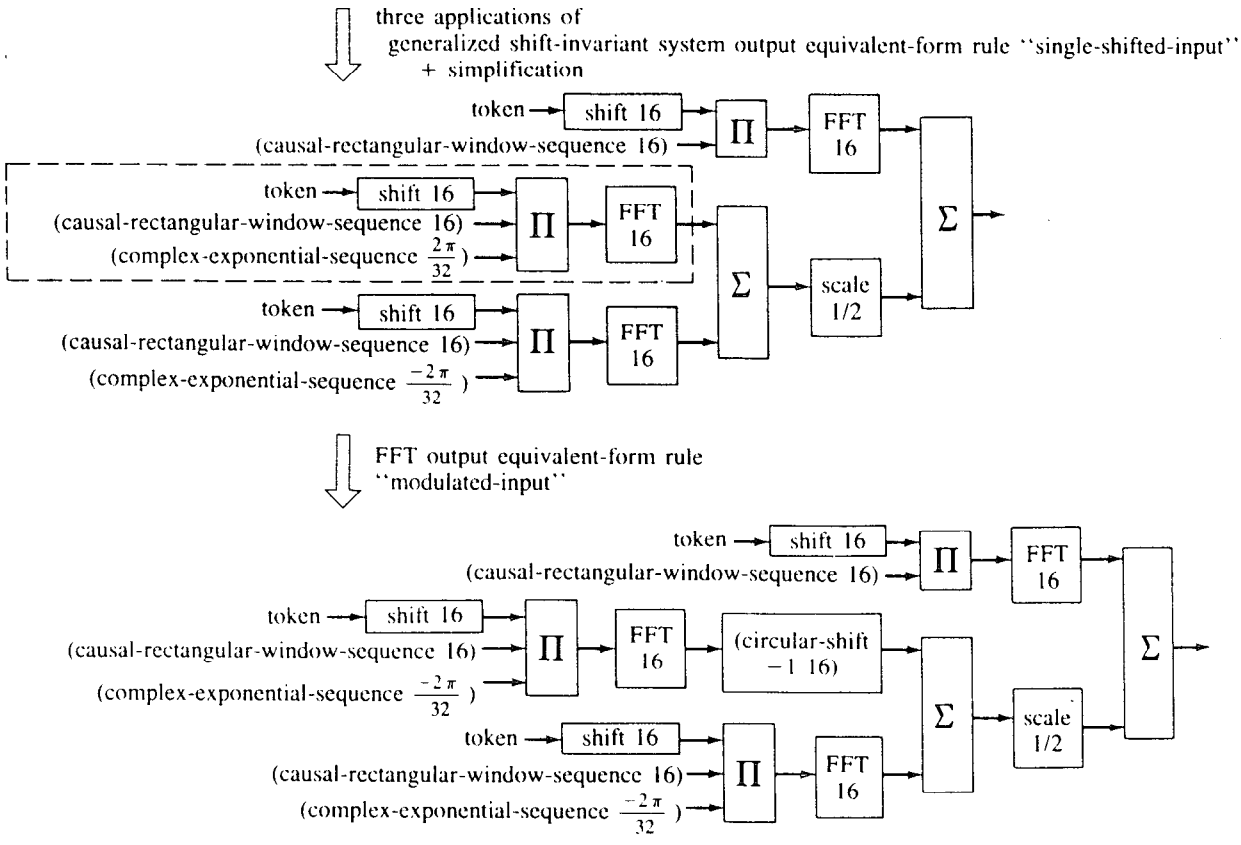
two applications of
commutative, associative system output equivalent-form rule "self-application"
+ simplification



two applications of
additive-system output equivalent-form rule "added-inputs"
+ two applications of
additive-system output equivalent-form rule "shifted-added-inputs"
with simplification between



three applications of
generalized shift-invariant system output equivalent-form rule "single-shifted-input"
+ simplification


three applications of
generalized shift-invariant system output equivalent-form rule "single-shifted-input"
+ simplification
token
shift 16
(causal-rectangular-window-sequence 16)
Π
FFT 16
Σ
(complex-exponential-sequence $\frac{2\pi}{32}$)
(complex-exponential-sequence $\frac{-2\pi}{32}$)
scale 1/2
FFT output equivalent-form rule
"modulated-input"
(circular-shift −1 16)

SYMBOLIC AND KNOWLEDGE-BASED SIGNAL PROCESSING

EDITORS

Alan V. Oppenheim

Massachusetts Institute of Technology

S. Hamid Nawab

Boston University

Prentice Hall

Englewood Cliffs, New Jersey 07632